Homotopy and Concurrency

Jeremy Gunawardena
BRIMS, Hewlett-Packard Labs
Filton Road, Stoke Gifford
Bristol BS12 6QZ, UK
jhcg@hplb.hpl.hp.com

September 13, 1994

appearing in the
Bulletin of the EATCS, 54, 184-193, October 94

Abstract

In this paper we give a homotopy theoretic proof of a well-known result in database engineering: that 2-phase locking is safe. The proof gives an immediate intuitive reason for why the 2-phase locking condition implies safety. We point out a number of interesting open questions regarding the interplay between homotopy and concurrency.

Keywords: Homotopy theory, serializability, 2-phase locking, concurrency theory

1 Introduction

What has homotopy got to do with concurrency? At first sight it seems unlikely that there should be any relationship between these two subjects. After all, homotopy theory is about continuous objects while concurrency typically deals with discrete structures. In this paper I will try to show that, on the contrary, there may be a very natural relationship between the two.

Instead of making a lot of abstract statements about homotopy and concurrency I would like to work through the proof of a theorem which all database engineers learn at their Mother's knee: that "two phase locking is safe". This result has been of considerable practical significance and still remains an important weapon in the database engineer's arsenal. The proof which I shall give is very simple and only makes use of homotopy in the most elementary way but it does give us a concrete example of the potential relationship between homotopy and concurrency.

In recent years a number of people have used ideas from homotopy theory and algebraic topology to study concurrency. Vaughan Pratt, [15], Rob van Glabbeek, [17], Eric Goubault and Thomas Jensen, [8, 9], have developed a theory of "higher dimensional automata" as a semantic framework for true concurrency. In a different direction, Soma Chaudhuri, [4], Michael Saks and Fotios Zaharoglu, [16], Elizabeth Borowsky and Eli Gafni, [2], Maurice Herlihy and Nir Shavit, [10, 11, 12], have found important generalisations of the classical Fischer, Lynch and Patterson Theorem, [7], using arguments from combinatorial and algebraic topology. Serious discussion of these contributions would exceed my brief here. This paper is an appetizer; the main course awaits those who are tempted to follow the pointers.

2 Two phase locking is safe

Imagine a centralized database acted upon concurrently by a finite number of transactions. In order to ensure that different transactions do not attempt to update the same record at the same time, each record is protected by a lock, or semaphore. A transaction which wishes to access a record must first acquire the lock and by doing so will lock out the others. We shall assume that each transaction consists of a sequence of record accesses known in advance. Of course a transaction might do some complicated operations on the data which it accesses but we are only interested in its interaction with the database and not on the details of what it does with the data. Using Djikstra's P, V notation for semaphores, [6, §3.2], we can conveniently write down a transaction in the form of a string like

$$PaPbVaPcVcVb$$

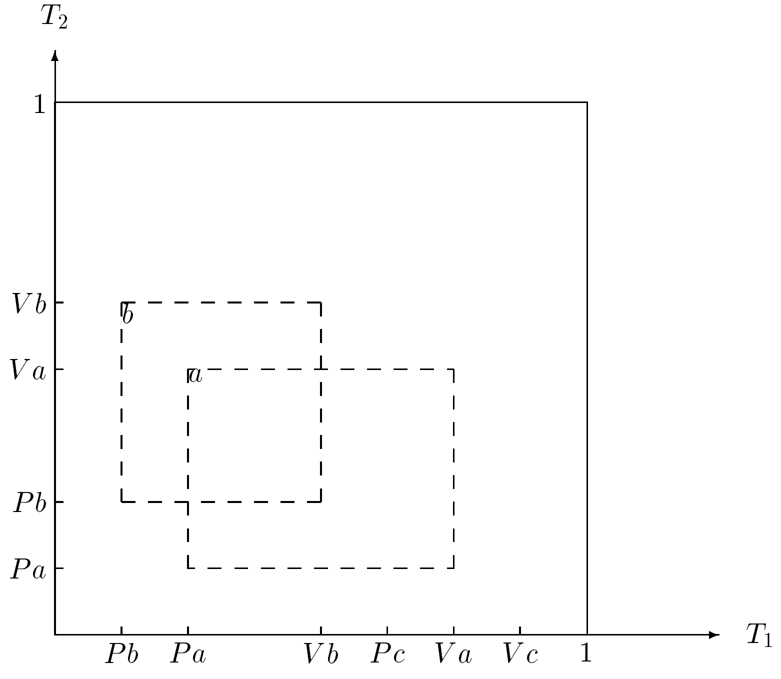
where Pa denotes the acquisition of the lock for record a and Va denotes the act of relinquishing it. Note that in any prefix of such a string, the difference between the number of occurrences of Pa and the number of occurrences of Va, is either 0 or 1. For our purposes we can assume that a transaction only accesses a record at most once.

The execution of a set of transactions is modelled by some interleaving of the individual P and V actions in each transaction. This is standard interleaving semantics and you can easily write down a CCS expression which captures the behaviour. Alternatively, you can imagine a scheduler sitting between the transactions and the database and choosing non-deterministically at every step which transaction should go next.

There is a simple geometric way of representing such semaphore programmes which goes back to Djikstra. Think of the sequence of P and V actions in a transaction as being represented by a sequence of real numbers. Which sequence of real numbers does not really matter. The important point is that the ordering of P and V actions in the transaction is faithfully captured by the numerical ordering on the corresponding numbers. It is convenient to assume that each transaction starts at the number 0, then performs all its record accesses and finally finishes at the number 1. Hence each P and V action is represented by some real number in the interval $(0, 1)$. If there are n transactions, $T_1, \cdots, T_n$ then we can think of T_i as being represented by the i-th coordinate axis in Euclidean space $\mathbf{R}^n$. The state of the system at any time is then represented by a point in $\mathbf{R}^n$ whose coordinates correspond to the actions which have just been executed by each transaction. The system consisting of the two transactions

$$\begin{aligned} T_1 &= PbPaVbPcVaVc \\ T_2 &= PaPbVaVb \end{aligned}$$

can be represented by the geometric figure below.



(1)

The big square box in the picture represents the unit cube $\{\vec{x} \in \mathbf{R}^n \mid 0 \leq x_i \leq 1\}$. The time evolution of the system describes a path leading from the origin, $(0, \cdots, 0)$, to the point $(1, \cdots, 1)$. Such execution paths are constrained to always increase in each coordinate; time cannot go backwards. They also have a staircase structure in that they consist only of linear segments parallel to the co-ordinate axes. Each segment corresponds to the execution of one of the transactions.

The locks on the records delineate certain "forbidden" regions of space which the system cannot enter. In two dimensions these forbidden regions take the form of boxes, or rectangles. The example above shows two such boxes, indicated in "dashed" outline, corresponding to the two records, a and b, which are accessed by both transactions. The record c which is accessed by T_1 but not by T_2 does not give rise to a forbidden region, for obvious reasons.

The execution paths of the system must lie outside the forbidden regions. Of course, it is possible for an execution path to get itself into a situation from which it cannot progress further. For instance, the two forbidden regions in (1) demarcate a corner, with coordinates (Pa, Pb), which will trap any path that goes diagonally outwards from the origin. Once the path has reached this corner it cannot progress without either violating one of the forbidden boxes or running time backwards. This corresponds to a deadlock.

This geometric way of representing the behaviour of semaphore programmes seems to have first appeared in [5] where it is attributed, without reference, to Djikstra. It is sometimes referred to as a "progress graph". The problem of detecting deadlocks in such systems has generated some elegant work which exploits the underlying geometry; see, for instance, the paper by Carson and Reynolds, [3]. Semaphore programmes are more subtle than one might think at first sight, especially when there are more than two transactions. An useful example to think about is the following

$$\begin{array}{rcl} T_1 & = & PxPyPzVxPwVzVyVw \\ T_2 & = & PuPvPxVuPzVvVxVz \\ T_3 & = & PyPwVyPuVwPvVuVv \end{array} \qquad (2)$$

which is due to Lipski and Papadimitriou, [13, §5.3], and is discussed further by Carson and Reynolds, [3, Appendix]. This example is deadlock free but proving this is not easy.

In dimensions higher than two, the forbidden regions are no longer simple rectangular boxes. Drawing these regions even in dimension three is beyond my TEX expertise but it is quite easy to describe them. It will be helpful to use the standard partial order on vectors in $\mathbf{R}^n$ which comes from the product ordering on $\mathbf{R}$: $\vec{x} \le \vec{y}$ if $x_i \le y_i$ for all $1 \le i \le n$. If $\vec{x} \le \vec{y}$ then the box with corners $\vec{x}$ and $\vec{y}$ is just the region $\{\vec{v} \mid \vec{x} \le \vec{v} \le \vec{y}\}$. Forbidden regions are made up of such boxes in the following way. Suppose that we have n transactions $T_1, \cdots, T_n$ and that a is a record which is accessed by some subset of transactions whose subscripts lie in the set $S \subseteq \{1, \cdots, n\}$. For each pair $i, j \in S$ with $i \ne j$, consider the box with corners $\vec{x}$ and $\vec{y}$ where

$$x_k = \begin{cases} 0 & \text{if } k \notin \{i,j\} \\ Pa & \text{otherwise} \end{cases} \qquad \text{and} \qquad y_k = \begin{cases} 1 & \text{if } k \notin \{i,j\} \\ Va & \text{otherwise.} \end{cases}$$

Clearly, $\vec{x} < \vec{y}$ since no transaction can relinquish a lock without first acquiring it. The forbidden region corresponding to the record a is then the union of such boxes obtained from all possible choices of distinct $i, j \in S$. There are hence $p(p-1)/2$ boxes making up each forbidden region, where p is the cardinality of S.

In dimension two there is only one box for each shared record, as in (1) above and the forbidden regions are convex. This is no longer the case in higher dimensions but the forbidden regions are, at least, "star shaped". To make this precise let us use the same notation as in the previous paragraph and consider the box with corners $\vec{a}$ and $\vec{b}$ where

$$a_k = \begin{cases} 0 & \text{if } k \notin S \\ Pa & \text{otherwise} \end{cases} \qquad \text{and} \qquad b_k = \begin{cases} 1 & \text{if } k \notin S \\ Va & \text{otherwise.} \end{cases}$$

It is easy to see that this box is contained in each box which makes up the forbidden region. I shall call this box the "centre" of the forbidden region. It has the following simple property. Suppose that $\vec{u}$ is in the centre of some forbidden region and that $\vec{v}$ is also in the same forbidden region. Then the line segment joining $\vec{u}$ to $\vec{v}$ lies entirely in the forbidden region. This is easy to see since $\vec{v}$ must lie in one of the boxes which make up the forbidden region and any box is convex. This is what I mean by "star shaped".

So far I have not explained what two phase locking is about. Database engineers are much concerned with ensuring the consistency of a database in the face of multiple transactions accessing the same records for different purposes. A simple way of ensuring such consistency is to force the transactions to execute in series, one after the other. The trick is to allow some level of concurrency without compromising the consistency. A particular execution of a set of transactions is said to be serializable if it has the same effect on the database as some serial execution of the same transactions. A transaction system is said to be safe if any non-deadlocked execution of it is serializable.

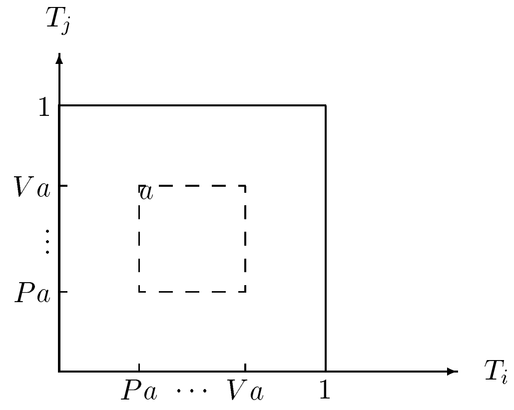
It is important to be clear about what it means for two transactions to "have the same effect". Suppose that record a is a single variable which happens to have the value 2 and that transaction T_1 is the operation $a := a + 2$ while T_2 is the operation $a := 2a$. Well, T_1 and T_2 certainly have the same effect on the record a; the value of a is 4 no matter which transaction is applied. However, this is an accident of the particular state of the database and the particular operations in the transactions. What is meant by "have the same effect" is that the database should be the same independently of its initial state and of the particular operations in the transactions.

Two phase locking is a rule for ensuring safety. It is very simple to state: each transaction must acquire all its locks before relinquishing any. In other words each transaction goes through two phases: in the first phase, the growing phase, it acquires all its locks and in the second phase, the shrinking phase, it gives them all back. What happens within each phase is unimportant and locks can be acquired and given up in any order. Two phase locking is a popular, albeit rather venerable, method; see [1, Chapter 3] and the references cited therein for further information.

The transaction system in (1) is not two phase locked since T_1 acquires the lock for record c after relinquishing the lock for record b. However, since the record c is not shared, it is not hard to see that the system is still safe. 2PL is not a necessary condition for safety. The system (2) is also not two phase locked but this example is not safe, as Lipski and Papadimitriou point out, [13, §5.3]. One of the basic results in database theory is that "2PL is safe": any transaction system which is two phase locked is necessarily safe. Note that we are not concerned with deadlock problems. The transaction system in (1) certainly has a deadlock. It is up to the scheduler either to detect this in advance and to schedule the transactions so as to avoid it or to wait for the deadlock to occur and roll back the execution (ie: to run time backwards). Deadlock detection and recovery are important issues for database engineers but they do not concern us here; see [1] for more information.

Why does 2PL work? It is at this point that ideas from topology become useful. Consider two paths of a transaction system which are homotopic. To make this precise, think of a path as a continuous function, $\alpha : I \to \mathbf{R}^n$, where $I = [0, 1]$ is the closed unit interval. Two paths, α and β, are said to be homotopic if there exists a continuous function $F : I \times I \to \mathbf{R}^n$ such that $F(x, 0) = \alpha(x)$ and $F(x, 1) = \beta(x)$ for all $x \in I$. All our homotopies will keep the end points fixed: $F(0, t) = F(0, 0) = F(1, 0)$ and $F(1, t) = F(1, 0) = F(1, 1)$ for all $t \in I$. Suppose then that α and β are execution paths (ie: paths which correspond to actual executions of the system) and that α and β are homotopic through a homotopy which does not encroach on any forbidden region. That is, $F(I \times I)$ does not intersect any forbidden region. In this case the two executions must have exactly the same eventual effect on the database. Why? Since the homotopy does not cross a forbidden region, I claim that it cannot alter the order in which the transactions access a shared record. If this is the case then clearly each execution must have the same eventual effect on the database.

The claim requires a little extra justification. To see why it must be so, assume that T_i and T_j are two transaction which both access record a. Assume further, without loss of generality, that $i < j$. Consider what we get when we project the transaction system onto the i and j axes. That is, consider the projection $\pi : \mathbf{R}^n \to \mathbf{R}^2$ given by $\pi(\vec{x}) = (x_i, x_j)$. Note that π is monotonic. The two execution paths project to two execution paths from $(0,0)$ to $(1,1)$. The original homotopy between the paths in $\mathbf{R}^n$ also projects to a homotopy in $\mathbf{R}^2$. This homotopy must avoid the box in the unit square with corners (Pa, Pa) and (Va, Va), for if not, the original homotopy would intersect the forbidden region for record a. (Why?) But then it must be the case that the projected paths both go around "on the same side" of the 2-dimensional box with corners (Pa, Pa) and (Va, Va). The situation is illustrated below. Note the importance of keeping the end points fixed during the homotopy.



It follows that T_i and T_j must access a in the same order on both execution paths: either T_i acquires the lock for a first on both paths (if the paths go below and to the right) or T_j acquires the lock first on both paths (if the paths go to the left and above). Since this is true for all pairs of transactions and all shared records, the claim made above is justified.

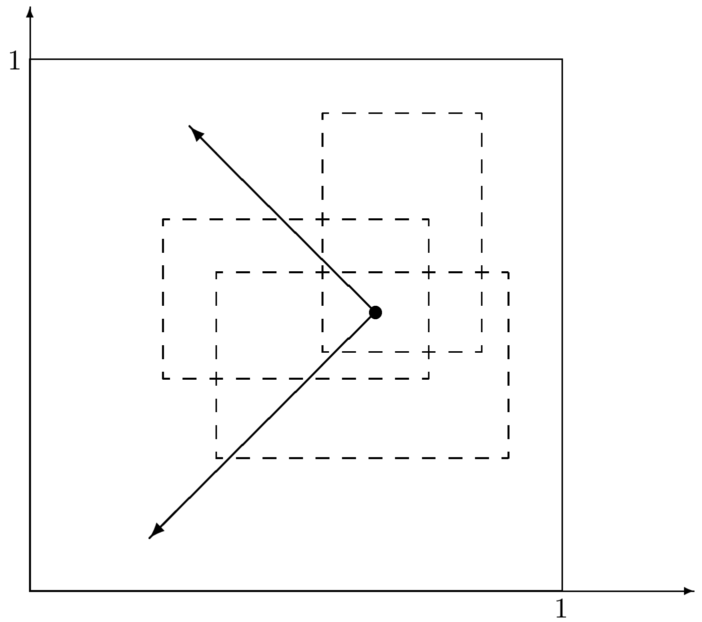
Serializability requires that a path have the same effect as some serial execution of the transactions. The serial executions correspond to going around the one dimensional "boundary", or "1-skeleton of the cubical complex" as a topologist would say, executing one transaction completely before going on to another. We can now begin to understand the significance of two phase locking. Let c_i denote a number on the i-th coordinate axis that lies between the two phases of the corresponding transaction. That is, it occurs after all the P actions and before any of the V actions. Such a number can always be found by virtue of the two phase locking condition. It is easy to see that the point with coordinates $(c_1, \cdots, c_n)$ must lie in the centre of each forbidden region. But now consider a radial projection from this point onto the boundary of the unit cube. Because of the "star shaped" property of forbidden regions, which we described above, this radial projection provides a homotopy, keeping the end points fixed, which carries any non-deadlocked execution path into the boundary of the unit cube. You just slide the path along the ray away from $\vec{c}$ until the path reaches the boundary; see the picture in (3) below.

One point to bear in mind is that this sliding does not necessarily give rise to a path which represents an execution: the path may no longer be composed of steps parallel to the coordinate axes. It is at this point that we make use of the non-discrete nature of the underlying topological space.

Once the path is in the boundary of the unit cube, it can be broken up into a number of segments each of which lies in a face of the cube. These segments no longer start and end in the corners but this will not affect the remainder of the argument. Each face of the cube corresponds to a transaction system obtained from $T_1, \cdots, T_n$ by omitting one of the transactions, say T_k, either before T_k has started (for the face containing points with $x_k = 0$) or after T_k has finished (for the face containing points with $x_k = 1$). We can repeat the argument using as a centre for the radial projection either

$$(c_1, \cdots, c_{k-1}, 0, c_{k+1}, \cdots, c_n) \qquad \text{or} \qquad (c_1, \cdots, c_{k-1}, 1, c_{k+1}, \cdots, c_n)$$

according to which face we are in. Proceeding in this way by iterated radial projections, we end up eventually in the 2-skeleton with faces of dimension two, corresponding to some pair of transactions from the original set. The forbidden regions in this case are convex and it is particularly easy to see how the radial homotopy works. We have sketched it below.



(3)

Any path segment is moved outwards by the homotopy into the 1-skeleton. Note that the points on the boundary are left fixed by the radial projection. It follows that we can fit together all the various segments to end up, finally, with a path that stays within the 1-skeleton.

At this point we are essentially finished. However, we have to pay the price for our temerity in using continuous methods. The path that we finally end up with may not represent a valid execution of the system. Suppose that α is our original path and that ω is the final path at the end of the iterated radial projections. It is quite possible that $\omega(t)$, as t runs from 0 to 1, goes along an edge in a negative direction, corresponding to time running backwards. It is instructive to imagine some staircase execution path superimposed on the unit square in (3) and to work out what happens to it under the radial projection. You will quickly observe that $\omega(t)$ can wobble about all over the place. How do we get around this?

Choose one of the coordinate axes, say i. We are going to observe what happens to $\omega(t)$ when we project the path onto this axis. Let $\pi : \mathbf{R}^n \to \mathbf{R}$ be the projection $\pi\vec{x} = x_i$. (I will omit brackets from the argument of π for clarity.) What can we say about $\pi\omega(t)$? The path ω is obtained by successive radial projections in various faces of the unit n-cube. These projections have the following nice property. If $\vec{y}$ is obtained from $\vec{x}$ by radial projection from $\vec{u}$, then $\vec{y}$, $\vec{x}$ and $\vec{u}$ lie on a straight line in the order in which I have just written them. But π is a linear and orientation preserving function. It follows that $\pi\vec{y}$, $\pi\vec{x}$ and $\pi\vec{u}$ must also lie on a straight line (which is obvious!) and in the same order (which is the important point). Since the centres of the radial projections are chosen consistently, we deduce the following:

$$\pi\alpha(t) \leq c_i \Longrightarrow \pi\omega(t) \leq \pi\alpha(t) \qquad \text{and} \qquad \pi\alpha(t) \geq c_i \Longrightarrow \pi\omega(t) \geq \pi\alpha(t).$$

We can now work out what happens to $\pi\omega$. It can oscillate for a long time in the interval $[0, c_i)$. It must eventually go through the point c_i, since, after all, $\pi\omega$ must eventually reach 1. However, once it has passed c_i, it can never return because the original path α, being monotonic, never does so. The same behaviour occurs in each direction $i \in \{1, \cdots, n\}$. It is now easy to see that by a homotopy, still keeping the end points fixed, we can move ω to remove all the "wobbles". We are left with a path which corresponds to a *bona fide* serialized execution. We deduce that 2PL is safe.

3 Discussion of the proof

I have omitted some of the details in the above proof but the reader should have no difficulty in filling in the gaps. I have not seen it published anywhere but that may well be due to my ignorance of the literature. I expect it is now well known, at least among the experts cited below. My apologies in advance to anyone whom I fail to mention.

The fact that homotopy of paths implies equivalence (in the sense that of having the same effect on the database) was first pointed out by Yannakakis, Papadimitriou and Kung, but only for systems in dimension 2, [19] and [13, Theorem 1]. They also gave a characterisation of safety in dimension 2 by requiring that a "convex closure"[1] of the forbidden regions be connected, [19, Theorem 3]. This characterisation does not extend to higher dimensions because connectedness of the closure no longer guarantees safety, [13, §5.3]; indeed, (2) is a counterexample. In dimension 2, the 2PL condition forces the forbidden regions to have a connected closure and this gives a geometric intuition for why 2PL works, [19, §4]. However, this intuition fails in higher dimensions. As we have seen, a better intuition is that 2PL provides a common point from which a radial homotopy is possible. This works in all dimensions.

The characterisation of safety in dimension 2 was later used by Lipski and Papadimitriou to give an elegant $O(n \log n \log\log n)$ algorithm for checking safety in two transaction systems with n records, [13].

The paper by Yannakakis, Papadimitriou and Kung contains much more. They give a characterisation of safety in any dimension and use this to prove not only that 2PL is safe (in any dimension) but also that other forms of locking, such as Tree Locking, are safe. However, homotopy plays no role in this.

It is interesting to speculate on whether a homotopical characterisation of safety is possible. The proof in §2 establishes rather more than I stated. Consider the complement of the forbidden

[1] This is not the convex closure commonly used in mathematics. See [13] or [19] for precise details.

regions within the unit cube. In general, this topological space will not be connected although it will have a finite number of components. Let C denote the component which contains the origin. C must contain the 1-skeleton since the origin is in the 1-skeleton and no part of the 1-skeleton can be in a forbidden region. Otherwise, the corresponding transaction would have a hard time making progress! The proof in §2 establishes that C deformation retracts onto the 1-skeleton. We recall that if $i : A \subseteq X$ is the inclusion of a subspace then X is said to deformation retract onto A if there exists a continuous function $p : X \to A$ such that pi is homotopy equivalent to 1_A, the identity function on A. The inductive argument given in §2 made use of successive radial homotopies and these can be patched together to give such a deformation retraction. Note that this is not any old deformation retraction: it has further properties which were important to the final proof of safety. Does this give us a characterisation of safety? In other words, is safety equivalent to the 1-skeleton being a specialised deformation retract of C?

Let me know if you find out.

It is a pity that the proof in §2 is not more widely available because it provides an immediate intuitive understanding of why 2PL works. I was once approached by a colleague at Hewlett-Packard who had been implementing a database system. He had been reading about locking mechanisms and serializability but had had a hard time understanding the theory behind two phase locking. When he saw the proof above he said "But that's obvious!" and went away muttering disgustedly about people who made things more difficult than they had to be.

There are good reasons for this difficulty. The topological ideas above work nicely on this particular problem but it is hard to see how to export them to other situations. A standard way of deducing the two phase locking theorem is given in [1, §3.3]. It is based on the Serializability Theorem, [1, Theorem 2.1], which gives a necessary and sufficient condition for a single execution to be serializable. This is analogous to the homotopy of paths condition established in §2. I should point out that modern database theory has moved on considerably from simple-minded transaction systems like those described above. Nowadays one must contend with distributed databases, nested transactions, multiple versions, replication, recovery from aborts, etc. The recent book by Lynch, Merritt, Weihl and Fekete, [14], gives an account of the modern approach which is based on the Atomicity Theorem, [14, Theorem 5.24], a substantial generalisation of the classical Serializability Theorem.

Is there a homotopy theoretic analogue of the Atomicity Theorem?

The proof of 2PL in §2 illustrates how topological ideas can be used to reason about concurrency. The state of a system is represented by a point in some topological space; an execution of a system by some path in the same space. Non-determinism is catered for by the possibility of several paths from initial state to final state. Homotopy indicates when executions are equivalent in some appropriate sense and the topology of the space dictates the properties and behaviour of the system. The spatial qualities of this representation arise from the commutativity of independent actions. (To put it another way, the holes in the space arise from mutual exclusion.) Commutativity has appeared in various guises in concurrency. To give just one example, Valmari, Godefroid, Wolper and others, [18], have used it to circumvent the state explosion problem. Commutativity is applied at each state, or locally, as a topologist would say, to show that certain paths need not be searched (because they are equivalent, in some sense, to paths which are searched). It is tempting to think of this as a form of homotopical collapsing, leading to an equivalent, but smaller, topological space.

Can this be made precise?

The possibility of a comprehensive theory of homotopy and concurrency is very attractive, at least to me. (The fact that I was an algebraic topologist before I got interested in concurrency has absolutely no bearing on this.) The spatial representation of behaviour is very intuitive, as we have seen. Admittedly, this intuition falls off quickly with increasing dimension. However, geometrical thinking has often shown the way forward when analysis and algebra (and logic?) have proved inadequate to the task. Poincaré, the father of algebraic topology, was deeply influenced to develop "qualitative methods" by the intransigence of nonlinear differential equations to the standard methods of analysis. Since Poincaré's time a large number of powerful mathematical tools have been developed to study homotopical properties. It would be tremendously exciting if these tools could be put to use to prove deeper results about concurrency. We might be able to give my engineering colleague at Hewlett-Packard a few more theorems like "2PL is safe" to carry around with him as mental building blocks for designing computer systems. The work of Herlihy *et al*, which I cited in the Introduction, suggests that this possibility is not as far-fetched, nor as far away, as one might have thought.

Acknowledgments

A number of people suggested that I write up the 2PL proof after seeing it sketched on tablecloths and scraps of paper at various venues. I am very grateful to Mogens Nielsen for providing a suitable opportunity for doing so and for putting up with my homotopical attitude towards deadlines (stretched but not broken!). My thanks also to Eric Goubault for many stimulating conversations and to Maurice Herlihy for sending me copies of his latest papers.

References

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

[2] E. Borowsky and E. Gafni. Generalized FLP impossibility result for t-resilient asynchronous computation. In *Proceedings 25th Annual ACM STOC*, 1993.

[3] S. D. Carson and P. F. Reynolds. The geometry of semaphore programs. *ACM TOPLAS*, 9(1):25–53, 1987.

[4] S. Chaudhuri. More choices allow more faults: set consensus problems in totally asynchronous systems. *Information and Computation*, 105:132–158, 1993.

[5] E. G. Coffman, M. J. Elphick, and A. Shoshani. System deadlocks. *ACM Computing Surveys*, 3(2):67–78, 1971.

[6] E. W. Djikstra. Co-operating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–112. Academic Press, 1968.

[7] M. J. Fischer, N. A. Lynch, and M. S. Patterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32:374–382, 1985.

[8] E. Goubault. Domains of higher dimensional automata. In E. Best, editor, *CONCUR'93*. Springer LNCS 715, 1993.

[9] E. Goubault and T. P. Jensen. Homology of higher dimensional automata. In W. R. Cleaveland, editor, *CONCUR'92*. Springer LNCS 630, 1992.

[10] M. Herlihy. A tutorial on algebriac topology and distributed computation. Draft available from herlihy@crl.dec.com, 1994.

[11] M. Herlihy and N. Shavit. The asynchronous computability theorem for t-resilient tasks. In *Proceedings 25th Annual ACM STOC*, 1993.

[12] M. Herlihy and N. Shavit. A simple constructive computability theorem for wait-free computation. In *Proceedings 26th Annual ACM STOC*, 1994.

[13] W. Lipski and C. H. Papadimitriou. A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems. *Journal of Algorithms*, 2:211–226, 1981.

[14] N. Lynch, M. Merritt, W. Weihl, and A. Fekete. *Atomic Transactions*. Morgan Kaufmann, 1994.

[15] V. Pratt. Modeling concurrency with geometry. In *Proceedings 18th ACM Symposium on POPL*, 1991.

[16] M. Saks and F. Zaharoglou. Wait free k-set agreement is impossible: the topology of public knowledge. In *Proceedings 25th Annual ACM STOC*, 1993.

[17] R. van Glabbeek. Bisimulation semantics for higher dimensional automata. Draft available from rvg@cs.stanford.edu, 1991.

[18] P. Wolper and P. Godefroid. Partial order methods for temporal verification. In E. Best, editor, *CONCUR'93*. Springer LNCS 715, 1993.

[19] M. Yannakakis, C. H. Papadimitriou, and H. T. Kung. Locking policies: safety and freedom from deadlock. In *Proceedings 29th IEEE Symposium on FOCS*, pages 286–297, 1979.