Pre–Analysis Locking: A Safe and Deadlock Free Locking Policy

Georg Lausen, Eljas Soisalon-Soininen[1], Peter Widmayer

Institut für Angewandte Informatik und Formale Beschreibungsverfahren
Universität Karlsruhe, Postfach 6380, 7500 Karlsruhe, West Germany

Abstract

A safe and deadlock free lock policy is introduced, called pre–analysis locking. Pre–analysis locking is based on an efficient geometric algorithm which inserts lock and unlock operations into the transactions. Pre–analysis locking is the first safe and deadlock free general locking policy which is not a variant of two–phase locking. It is an approach conceptually different from policies following the two–phase locking principle. In general, none of pre–analysis locking and two–phase locking dominates the other: there exist cases in which pre–analysis locking allows for more concurrency than any two–phase locking policy, but there are also cases in which a two–phase locking policy allows for more concurrency than pre–analysis locking.

Keywords:

database concurrency control, locking policy, serializability, safety, deadlock

1. Introduction

A database consists of a set of entities describing the application of relevance. Transactions are user processes which transform a consistent state of a database into a new consistent state. The measure taken to achieve consistency is a set of constraints on the entities defining the valid states of a given application environment. The calculations of a transaction are performed by executing several atomic read or write actions on the database. There may be some temporary violations of the consistency constraints during a transaction, but at the end all restrictions are fulfilled. Certainly, this consistency preservation property of a transaction is guaranteed if each transaction is executed alone on the database. For efficiency reasons, transactions should be executed concurrently. Lock policies are a widely used mechanism to coordinate a concurrent set of transactions in a way which guarantees a consistent view on the database for each process, i.e., which guarantees safety. Safety is enforced if the lock policy allows only serializable schedules. A schedule is an interleaving of the actions of the transactions. For a serializable schedule, the computations performed by all transactions are equivalent to a not interleaved, namely serial, schedule of the same transactions.

Locking has been studied extensively in the past. In [4, 8, 16] it is shown that safe lock policies inherently cannot allow all possible serializable schedules, even if we restrict general serializability to D-serializability, which is an efficiently decidable restricted version of serializability, while the problem in general is NP–complete [7]. Besides safety, freedom from deadlock is another important property lock policies should have. Deadlocks are cyclic wait relationships between transactions, which can be resolved only by aborting at least one involved transaction.

Previous work on concrete lock policies can be classified according to whether or not a structure on the database is assumed, which restricts the allowed orderings of the lock operations in a transaction. For example, one familiar structure are trees — an entity may only be locked if its father currently is locked. Safe and deadlock free lock protocols in the case of structured databases are investigated

[1] The work of this author was supported by the Alexander von Humboldt Foundation. Present address: Department of Computer Science, University of Helsinki, Tukholmankatu 2, SF-00250 Helsinki 25, Finland.

exhaustively in [2, 3, 10, 11, 15]. If no structure on the set of entities is assumed, i.e., the transactions may lock the entities in any order, two-phase locking (2PL) is the protocol commonly being used [1]. In 2PL a transaction must lock every entity before it accesses it, but, after some entity is unlocked, no succeeding lock is allowed any more. Unfortunately, 2PL is not free from deadlock.

If no structure on the database is assumed, the only way to develop improvements over 2PL is to analyze the action sequences of the transactions.

Recently in [13] an interesting algorithm is proposed which inserts unlock operations into transactions, which initially contain only actions and lock operations. The resulting non-2PL lock policy is a heuristic for a safe improvement over 2PL, where the measure of concurrency is related to the time entities are being kept locked.

Freedom from deadlock for 2PL can be achieved by a simple analysis called preclaiming [12]: one lock operation locking all entities accessed by the transaction is executed as the transaction's first step. Obviously, preclaiming reduces potential concurrency, since locks have to be acquired earlier than it would be necessary for 2PL in general.

Further, there most probably does not exist an efficient strategy to transform an initial set of 2PL transactions into a set of deadlock free 2PL transactions without reducing potential concurrency unnecessarily. It is shown in [15] that testing a set of 2PL transactions for freedom from deadlock is NP-complete.

In this paper we further investigate the question of safe and deadlock free policies in the case of an unstructured database. A non-2PL safe lock policy is introduced, called pre-analysis locking (PAL). PAL is based on an efficient geometric algorithm which inserts lock and unlock operations into the action sequences of the transactions. Even though lock operations need not be acquired at the beginning of each transaction, PAL is a policy free from deadlock. To the authors' knowledge, PAL is the only safe and deadlock free general policy known today which is not a variant of 2PL.

PAL and 2PL policies are conceptually different approaches. In 2PL policies lock and unlock operations are related to entities — in PAL lock and unlock operations are only syntactic operations. PAL and 2PL do not strictly dominate each other in general: there exist schedules which are allowed by PAL, but by no 2PL policy, and, on the other hand, there exist schedules which are allowed by preclaiming 2PL, but not by PAL. An exact characterization of PAL with respect to 2PL policies is an interesting open research area.

The paper is organized as follows. In the next section, we present the definitions and facts necessary in the sequel. Section 3 illustrates concurrency in a geometric setting, and Section 4 presents and analyzes PAL in some detail. Section 5 shows the freedom from deadlock of PAL, and Section 6 discusses implications of the results.

2. Basic Definitions and Facts

A **transaction system** $\tau = \{T_1, \ldots, T_d\}$ is a set of **transactions**. A transaction $T_i = (T_{i1}, \ldots, T_{im})$ is a sequence of **actions**. Each action T_{ij} has associated with it an **entity** $x_{ij} \in E$, where E is a set of entities forming the database. We distinguish read and write actions, $T_{ij} = R_{ij}$ meaning **read** x_{ij} and $T_{ij} = W_{ij}$ meaning **write** x_{ij}. Each transaction reads each entity at most once and writes each entity at most once, and it is not allowed to read an entity after it has written it. A **schedule** s of τ is a permutation of all actions of τ such that $1 \leq j < k \leq m_i$ implies $s(T_{ij}) < s(T_{ik})$.

Two actions of two different transactions **conflict**, if they involve the same entity, and at least one of them is a write action. With each schedule s we associate a labelled directed graph $D(s)$ having as nodes the transactions of τ and an edge $T_i \xrightarrow{x} T_j$ for any action of T_i involving x that precedes in s a conflicting action of T_j. A schedule is called **D-serializable** (conflict serializable) if $D(s) = D(s')$ for some serial schedule s' of τ, or, equivalently, if $D(s)$ is acyclic. A transaction system τ is called **D-safe** if every schedule of τ is D-serializable.

For testing D-serializability and D-safety, conflicts between transactions must be considered. We define a binary relation on the set of actions of the transaction system in the following way. We say $p = (T_{iu}, T_{jv})$ is a **direct conflict point** between T_i and $T_j, i \neq j$, if T_{iu} and T_{jv} conflict. Point $p = (T_{iu}, T_{jv})$ is called a **conflict point** between T_i and T_j, $i \neq j$, if there exists a set of transactions $\{T_{k_l} | 1 \leq l \leq m\} \subseteq (\tau - \{T_i, T_j\})$ for some m, such that for all l, $1 \leq l \leq m-1, (T_{k_l p_l}, T_{k_{l+1} q_{l+1}})$ is a direct conflict point between T_{k_l} and $T_{k_{l+1}}$, and $(T_{iu}, T_{k_1 q_1})$ and $(T_{k_m p_m}, T_{jv})$ are direct conflict points between T_i and T_{k_1}, T_{k_m} and T_j, respectively. The sequence $(T_{iu}, T_{k_1 q_1})$, $(T_{k_1 p_1}, T_{k_2 q_2}), \ldots,$ $(T_{k_{m-1} p_{m-1}}, T_{k_m q_m})$, $(T_{k_m p_m}, T_{jv})$ is called a **conflict point path** for p. The number, m, of intermediate transactions is called the **length** of the path. In the case of $m = 0$, p is a direct conflict point; otherwise, i.e., for $m \geq 1, p$ is called an **indirect conflict point**.

3. The Geometry of Concurrency

In this section we will briefly mention a geometric characterization of concurrency, in the same spirit as locking has first been characterized in [9]. Let τ be a transaction system consisting of two transactions $T_1 = (T_{11}, \ldots, T_{1m_1})$ and $T_2 = (T_{21}, \ldots, T_{2m_2})$. In the coordinate plane, as illustrated in Figure 3.1, the two axes correspond to the transactions T_1 and T_2, and the integer points on the axes correspond to the actions in the transactions.

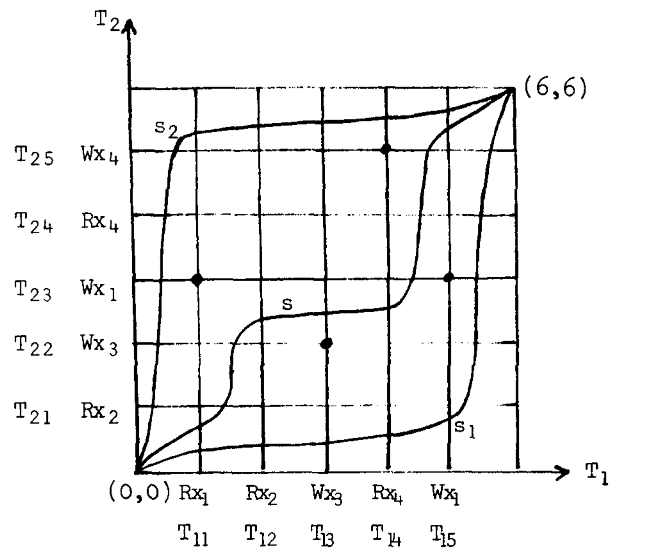


Figure 3.1: The representation of a two-transaction system

A schedule of $\{T_1, T_2\}$ has now the following geometric image: Any nondecreasing curve from point (0,0) to point (m_1+1, m_2+1) not passing through any other grid points represents the schedule where the actions $T_{11}, \ldots, T_{1m_1}, T_{21}, \ldots, T_{2m_2}$ are in the order in which the curve passes the lines $T_1 = T_{11}, \ldots, T_1 = T_{1m_1}, T_2 = T_{21}, \ldots, T_2 = T_{2m_2}$. As all curves representing a schedule are equivalent for our purposes, we will refer in the sequel to an arbitrary one of them, calling it the schedule's curve. For instance, in Figure 3.1, s_1 and s_2 correspond to the serial schedules $T_{11} \ldots T_{1m_1} T_{21} \ldots T_{2m_2}$ and $T_{21} \ldots T_{2m_2} T_{11} \ldots T_{1m_1}$, respectively, and s represents the schedule $T_{11}T_{21}T_{22}T_{12}T_{13}$ $T_{14}T_{23}T_{24}T_{25}T_{15}$. The conflict points in Figure 3.1 are (T_{11}, T_{23}), (T_{13}, T_{22}), (T_{14}, T_{25}), and (T_{15}, T_{23}). Clearly, the transaction system $\tau = \{T_1, T_2\}$ is not D-safe, since s is not a D-serializable schedule.

Let s be a schedule of a transaction system $\tau = \{T_1, T_2, \ldots, T_d\}$. For any pair $i, j \in \{1, \ldots, d\}, i \neq j$, the schedule s_{ij} is a schedule of the transaction system $\tau_{ij} = \{T_i, T_j\}$, and s_{ij} is derived as a projection from s by deleting all actions of transactions not in τ_{ij}.

Lemma 3.1: Let $\tau = \{T_1, T_2, \ldots, T_d\}$ be a transaction system, and let s be a schedule of τ which is not D-serializable. Then there exist transactions $T_i, T_j, i \neq j$, $i, j \in \{1, \ldots, d\}$, such that the curve of s_{ij} separates two conflict points in the plane (T_i, T_j) , at least one of which is a direct conflict point.

Proof: Because s is not D-serializable, $D(s)$ has a cycle, say $T_1 \rightarrow T_2 \rightarrow \ldots T_k \rightarrow T_1, k \geq 2$. Let T_{ip_i} and T_{jq_j}, respectively, be the actions of T_i and T_j causing the arc $T_i \rightarrow T_j$ in the cycle, $1 \leq i \leq k$, $j = i+1$ modulo k. This means that for any such i, j, $s(T_{ip_i}) < s(T_{jq_j})$, and (T_{ip_i}, T_{jq_j}) is a direct conflict point, and (T_{iq_i}, T_{jp_j}) is a conflict point between T_i and T_j. Assume for the sake of contradiction that for no such i, j, the curve of s_{ij} separates these two conflict points. Then for all i, j adjacent in the cycle, $s(T_{iq_i}) < s(T_{jp_j})$. But these two conditions impose a cyclic ordering on the actions T_{iq_i}, T_{ip_i} for all i, and hence no schedule s with its linear ordering of the actions can comply, a contradiction.

■

Since in general the complete set of D-serializable schedules cannot be achieved by locking [8, 16], our intention is to define a lock policy that rules out all those schedules which are not D-serializable, and basically in such a way that only very few D-serializable schedules will be ruled out. We will introduce a lock policy whose lock operation apply only to pairs of transactions. As noted in [8], this is no loss of generality. The lock operations are derived from forbidden regions in the planes corresponding to all pairs of transactions. The effect of a forbidden region on the set of allowed schedules can be seen by looking at the set of grid points (i.e., intersection points of grid lines) the forbidden region contains. The exact boundary of the forbidden region is irrelevant because a schedule is represented as a set of curves. Therefore, we restrict the subsequent discussion to rectilinear polygons, i.e. polygons with sides parallel to the coordinate axes, for the forbidden regions. To avoid ruling out schedules unnecessarily, such regions should be as small as possible while still guaranteeing safety. By Lemma 3.1 all schedules must be forbidden whose curves are separating conflict points in some plane. Therefore, the forbidden regions are connected **S(outh)W(est)-** and **N(orth)E(ast)-** closed regions, that implies, regions containing the south-west and north-east corner points of their smallest bounding rectangles. More formally, we say that a region R (a set of points in the plane) is **SW-closed**, if for any two points (x_1, y_1) and (x_2, y_2) in R, such that $x_1 < x_2$ and $y_1 > y_2$, also the point (x_1, y_2) is included in R. The **SW-closure** of region R, denoted $SW(R)$, is the smallest SW-closed region containing R. Analogously, we define that region R is **NE-closed**, if for anz two points (x_1, y_1) and (x_2, y_2) in R, such that $x_1 < x_2$ and $y_1 > y_2$, also the point (x_2, y_1) is included in R. The **NE-closure** of region R, denoted NE(R), is the smallest NE-closed region containing R. We say that region R is **NESW-closed**, if R is both SW- and NE-closed. The NESW-closure of R, denoted $NESW(R)$, is the smallest NESW-closed region containing R.

An example of a NESW-closed region is given in Figure 3.2.

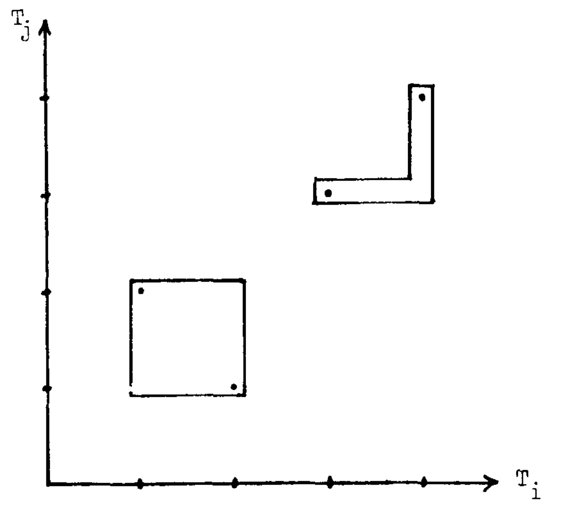


Figure 3.2: A NESW-closed region

Fact 3.2: Let $\tau = \{T_1, T_2\}$ be a transaction system, and let R be a connected region in the (T_1, T_2)-plane. The set of curves of schedules not cutting R equals the set of curves of schedules not cutting $NESW(R)$.

■

Region R in the plane is called **r-convex** (rectilinearly convex), if for any two points (x, y_1) and (x, y_2) in R the line segment between (x, y_1) and (x, y_2) is also included in R, and for any two points (x_1, y) and (x_2, y) in R, the line segment between (x_1, y) and (x_2, y) is also included in R. Given a region R in the plane, a **connected r-convex hull**, or **cr-convex hull**, of R, is a smallest connected r-convex region containing R. A **NESW-closed cr-convex hull** of a region R is the NESW-closure of a cr-convex hull of R. An example of different NESW-closed cr-convex hulls is given in Figure 3.3.

We say that a set of schedules of $\tau = \{T_1, T_2\}$ can be defined by a rectilinear forbidden area R in the plane (T_1, T_2), if this set is obtained by ruling out the schedules whose curves are intersecting R.

Fact 3.3: Let $\tau = \{T_1, T_2\}$ be a transaction system, and let some set of schedules of τ be defined by a forbidden area R in the (T_1, T_2)-plane. Then this set of schedules of τ can also be defined by a rectilinear forbidden area R' in the (T_1, T_2)-plane.

■

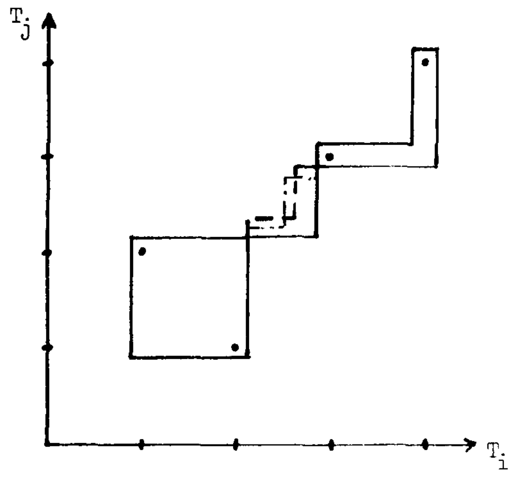


Figure 3.3: Three NESW-closed cr-convex hulls

4. The Pre-Analysis Lock Policy PAL

A **locked transaction system** $L\tau$ is a set of **locked transactions**, $L\tau = \{LT_1, LT_2, \ldots, LT_d\}$, where each locked transaction is a transaction containing **lock** v and **unlock** v operations besides the read and write actions, for $v \in LV$. LV is the set of **locking variables**; it is independent of the set of entities. For any $v \in LV$ and any locked transaction, there is at most one lock v and one unlock v operation in the transaction; lock v must be followed by unlock v, and unlock v must be preceded by lock v.

The meaning of the locks is that after the execution of operation lock v for some $v \in LV$ in locked transaction LT_i, operation lock v is not allowed to be executed in locked transaction $LT_j, j \neq i$, before unlock v has been executed in LT_i. The lock and unlock operations hence act as binary semaphores between transactions, and only schedules following these rules are allowed. We say that a locked schedule Ls of $L\tau$ is **legal**, if in Ls each lock v operation is followed first by an unlock v operation before the next lock v operation. $L(L\tau)$ denotes the set of **legal schedules** of $L\tau$. We say that $L\tau$ **realizes** the set of schedules $S\tau = L(L\tau)/\{\text{lock } v, \text{unlock } v | v \in LV\}$, where A/B denotes the deletion of all symbols in B from all words in A. $L\tau$ is D-safe, if all schedules in $S\tau$ are D-serializable.

$L\tau$ is called **free from deadlock (deadlock free)**, if every legal prefix of a locked schedule of $L\tau$ can be extended to a legal locked schedule of $L\tau$. A locked transaction system with a potential deadlock is undesirable because a schedule may start running correctly and get blocked later on in such a way that there is no correct completion.

Our goal is to develop a lock policy that allows us, for any given transaction system τ, to compute efficiently a safe and deadlock free locked version $L\tau$ of τ allowing for a high degree of parallelism. In our notion, the degree of concurrency is the number of legal schedules of a locked transaction system. We use Lemma 3.1 to define such a lock policy: We will forbid all schedules s , where for some i, j the curve of s_{ij} separates two conflict points in the plane (T_i, T_j), at least one of which is a direct one. The resulting locked transaction system is D–safe by Lemma 3.1. We will show in the next section that we construct $L\tau$ in such a way that it is also deadlock free. In general, we will have to rule out legal schedules as well by this method, but it is well known that a single locked transaction system must also have this property in general, even for a system of two transactions ([8, 14]).

This will be accomplished by constructing an NESW–closed cr–convex hull of the set of all conflict points in the plane (T_i, T_j), for each i, j with at least one direct conflict between T_i and T_j. More formally, the pre–analysis lock policy PAL for a given transaction system $\tau = \{T_1, T_2, \ldots, T_d\}$ operates as follows:

Algorithm Pre–Analysis Locking

```
1. Determine the set of direct and indirect conflict points
   between all pairs of transactions of $\tau$.

2. For each pair $T_i, T_j$ with a direct conflict point between
   $T_i$ and $T_j$, $i \neq j$, $1 \leq i,j \leq d$, construct a
   NESW–closed cr–convex hull of all conflict points in the plane $(T_i, T_j)$.

3. For each transaction $T_i$, $1 \leq i \leq d$, regard $T_i$ as a
   locked transaction $LT_i$ with no lock and unlock operations.

4. For each locked transaction $LT_i$, $1 \leq i \leq d$, do:
       For each transaction $LT_j$, $j \neq i$, $1 \leq j \leq d$,
       which has a direct conflict with $LT_i$, do:
           augment $LT_i$ by locks and unlocks derived from the
           NESW–closed cr–convex hull of the conflict points in the
           plane $(T_i, T_j)$; position directly adjacent locks
           according to a global order of all locking variables.

End of Algorithm
```

In order to demonstrate the efficiency of pre–analysis locking, we describe in some more detail how to carry out the operations of the algorithm.

First, all conflict points are being constructed. To this end, all direct conflict points for each pair T_i, T_j of transactions of τ are computed, $i \neq j, 1 \leq i,j \leq d$. Let $dp(i,j)$ be the set of direct conflict points between T_i and T_j, and let $dp_i(i,j)$ be the set of actions of T_i occurring in $dp(i,j)$, and let $dp_j(i,j)$ be the set of actions of T_j occurring in $dp(i,j)$. Construct an undirected graph $G = (V, E)$, where the set of vertices represents the set of transactions, denoted $V = \{T_1, \ldots, T_d\}$. There is an edge between T_i and T_j, denoted by (T_i, T_j), iff $dp(i,j) \neq \emptyset$. With each edge $(T_i, T_j) \in E$, associate the set $dp(i,j)$, for the time being. A simple path in G is a sequence of different edges $(T_{i_1}, T_{i_2}), (T_{i_2}, T_{i_3}), \ldots, (T_{i_{u-1}}, T_{i_u})$, with $u \geq 3$, such that for $w \neq w'$ $T_{i_w} \neq T_{i_{w'}}$; we say that the path connects T_{i_1} with T_{i_u}. The set of indirect conflict points between T_i and T_j, $i \neq j$, can be determined from the simple paths connecting T_i and T_j in the following way. Consider any such simple path, say the path $(T_i, T_k), \ldots, (T_h, T_j)$. Then for any action $x \in dp_i(i,k)$ and any action $y \in dp_j(h,j)$, point (x,y) is an indirect conflict point in the plane (T_i, T_j); the conflict point path is given by the simple path in G. Hence, all points $(x', y') \in dp_i(i,k) \times dp_j(h,j)$ are indirect conflict points between T_i and T_j, where $\times$ denotes the Cartesian product. However, for the construction of the NESW–closed cr–convex hull of the set of conflict points it is, of course, equivalent to consider the four corner points of the rectangular pattern of conflict points obtained by $dp_i(i,k) \times dp_j(h,j)$. Therefore, for the computation of the indirect conflict points it is sufficient to associate with each edge $(T_i, T_k) \in E$ the four actions $first_i(i,k)$, $last_i(i,k)$, $first_k(i,k)$, $last_k(i,k)$, instead of the entire set $dp(i,k)$, where $first_i(i,k)$ denotes the first of the actions of $dp_i(i,k)$ in T_i, and $last_i(i,k)$ denotes the last of the actions of $dp_i(i,k)$ in T_i, and similar for $dp_k(i,k)$ and T_k. The indirect conflict points between T_i and T_j resulting from simple path $(T_i, T_k), \ldots, (T_h, T_j)$ are then $(first_i(i,k), first_j(h,j)), (last_i(i,k), first_j(h,j))$, $(first_i(i,k), last_j(h,j))$, and $(last_i(i,k), last_j(h,j))$.

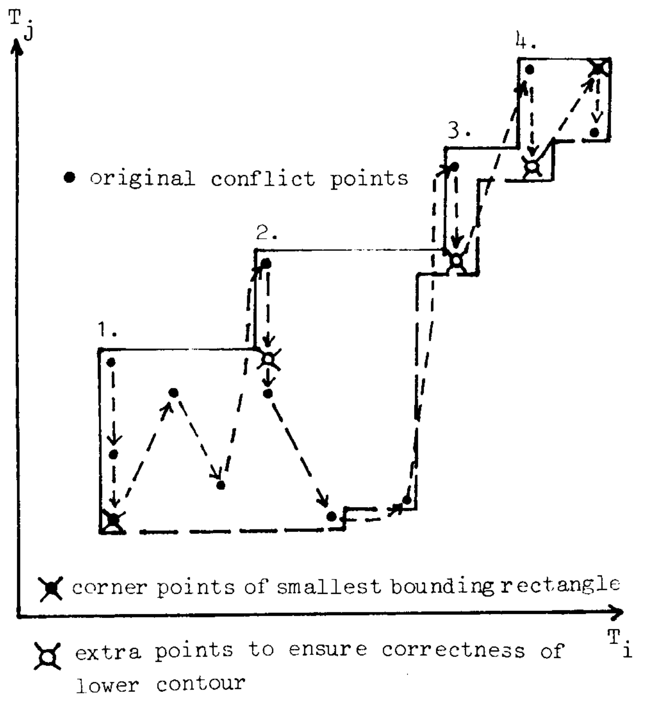
In order to determine all indirect conflict points it is prohibitive to explicitly investigate all simple paths. At first, because we are only interested in indirect conflict points for pairs of transactions having a direct conflict point as well, we remove from G all edges that do not belong to a (simple) cycle; note that the edge between the transactions in the considered pair forms a cycle with any simple path between one of the transactions and the other. We then compute the connected components of the modified graph, resulting in connected graphs $G_1, G_2, \ldots, G_c$, $c \geq 1$. For each $G_l = (V_l, E_l)$, $1 \leq l \leq c$, the set of edges is partitioned into equivalence classes $E_l^1, \ldots, E_l^r$, $r \geq 1$: two edges belong to the same class iff they are parts of a common simple cycle in G_l. From this definition it follows that there cannot exist a simple cycle whose edges belong to two different equivalence classes. Hence, when searching for all simple cycles, we may restrict our attention to edges within the same class E_l^s, $s \in \{1, \ldots, r\}$. Now consider the subgraph $G_l^s = (V_l^s, E_l^s)$, where V_l^s is the set of vertices occurring in E_l^s. Clearly, G_l^s is a connected graph. Let us focus on the computation of all indirect conflict points for a pair T_i, T_j of transactions, with $(T_i, T_j) \in E_l^s$. To this end, consider the graph $\bar{G}_l^s = (\bar{V}_l^s, \bar{E}_l^s)$, where $\bar{V}_l^s = V_l^s - \{T_i, T_j\}$, and $\bar{E}_l^s$ is obtained from E_l^s by removing all edges incident with T_i or T_j. Let $\bar{G}_l^s(1), \ldots, \bar{G}_l^s(f)$, $f \geq 1$, denote the connected components of $\bar{G}_l^s$, with $\bar{G}_l^s(g) = (\bar{V}_l^s(g), \bar{E}_l^s(g))$, for all $g \in \{1, \ldots, f\}$. By that definition, there does not exist a simple path from T_i to T_j in G_l^s containing two vertices from two different connected components of $\bar{G}_l^s$. Hence, any simple path from T_i to T_j in G_l^s has to be contained completely in some of the connected components of $\bar{G}_l^s$. For connected component $\bar{G}_l^s(g)$ of $\bar{G}_l^s$, consider the set of those vertices of $\bar{V}_l^s(g)$ that are incident with T_i in G_l^s, and let us denote them by $\bar{V}_l^s(g)_i$; analogously, let $\bar{V}_l^s(g)_j$ be the set of vertices of $\bar{V}_l^s(g)$ that are incident with T_j in G_l^s. Pick any two vertices T_k and T_h, with $T_k \in \bar{V}_l^s(g)_i$ and $T_h \in \bar{V}_l^s(g)_j$. Then, because T_k and T_h belong to $\bar{V}_l^s(g)$, there exists a simple path from T_k to T_h in $\bar{G}_l^s(g)$, and together with edges $(T_i, T_k) \in E_l^s$ and $(T_j, T_h) \in E_l^s$, this path determines a set of indirect conflict points between T_i and T_j, namely the points $\{first_i(i,k), last_i(i,k)\} \times \{first_j(h,j), last_j(h,j)\}$. As this observation holds for any pair of vertices picked from $\bar{V}_l^s(g)_i$ and $\bar{V}_l^s(g)_j$, we obtain the set $(\{first_i(i,k') | k' \in \bar{V}_l^s(g)_i\} \cup \{last_i(i,k') | k' \in \bar{V}_l^s(g)_i\}) \times (\{first_j(h',j) | h' \in \bar{V}_l^s(g)_j\} \cup \{last_j(h',j) | h' \in \bar{V}_l^s(g)_j\})$. For the purpose of constructing the NESW-closed cr–convex hull, again the four corner points of the rectangular pattern of indirect conflict points are sufficient.

Let us recall that for each connected component of graph $\bar{G}_l^s$, derived from G_l^s by removing vertices T_i and T_j, we obtain at most four indirect conflict points between T_i and T_j: we simply combine the actions of T_i and T_j associated with the edges between T_i and T_j and vertices in the respective component of $\bar{G}_l^s$. Because the number of components of $\bar{G}_l^s$ is limited by $O(d)$, so is the number of indirect conflict points between any pair of transactions, that need to be considered for the computation of the hull. Therefore, after the completion of Step 1 of algorithm PAL we are left with at most $O(nd)$ direct and $O(d^3)$ indirect conflict points, where n is the total number of actions of τ.

Second, taking the set of conflict points between T_i and T_j for each i, j as input, a NESW-closed cr-convex hull is constructed using an algorithm proposed in [6].

To this end, the left lower and the right upper corner points of the smallest rectangle bounding the set of conflict points is added to the set of conflict points. Then, these points are sorted lexikographically with increasing T_i–coordinate and decreasing T_j–coordinate (see Fig. 4.1). The sorted sequence of points is scanned, keeping track of all upper left corner points of a cr–convex hull of the points as follows. The first corner point is the first point in the sequence. As soon as during the scan a point with higher T_j–coordinate than the actual corner point is found, that points replaces the actual corner point. When all points have been scanned, all corner points have been found, that is, an upper contour of a NESW–closed cr–convex hull has been determined (see Fig. 4.1).

In case the NESW–closed cr–convex hull of the set of points is not uniquely defined, care must be taken to ensure that a lower contour does not intersect the upper contour. Therefore, the set of points considered for a lower contour is the preliminary set of points, augmented by the following: for any two corner points (x, y) and (x', y') that are adjacent on the upper contour and appear in this order during the scan, the point (x', y) is added to the set (see Fig. 4.1). Then, for the new set of points the lower contour is constructed similarly.



T_j
4.
3.
● original conflict points
2.
1.
corner points of smallest bounding rectangle
T_i
extra points to ensure correctness of lower contour

Figure 4.1: Construction of NESW–closed cr–convex hull

The third step, namely regarding the transactions as locked transactions, does not involve any algorithmic action.

Fourth, each NESW–closed cr–convex hull is covered exactly with a minimum number of (overlapping) rectangles in the following way, which was first described in [5]. Consider the sequence of corner points along the upper contour of hull H, in sorted order from left to right, say $p_1, p_2, \ldots, p_{up}$, and the sequence of corner points for the lower contour in the same order, say $q_1, q_2, \ldots, q_{low}$. Let $R(p_i, q_j)$ denote the rectangle with left upper corner point p_i and right lower corner point q_j, for q_j to the right and below p_i. Any such rectangle lies entirely in H, because the contours of H are nondecreasing in both coordinates.

Choose $R(p_1, q_1)$ to be the first rectangle to be used to cover H. Let $R(p_i, q_j)$ be the rectangle just chosen. Then the next rectangle is chosen as follows. If $p_i \neq p_{up}$ and $R(p_{i+1}, q_j)$ is a rectangle overlapping $R(p_i, q_j)$, then let $p' = p_{i+1}$, otherwise let $p' = p_i$. If $q_j \neq q_{low}$ and $R(p', q_{j+1})$ is a rectangle overlapping $R(p_i, q_j)$, then let $q' = q_{j+1}$, otherwise let $q' = q_j$. Now let $R(p', q')$ be the next rectangle to be chosen. This process is to be repeated until $p' = p_{up}$ and $q' = q_{low}$; then, H is covered by a minimum number of rectangles. (Note in this context that for the procedure to work it is essential that H is NE-closed.)

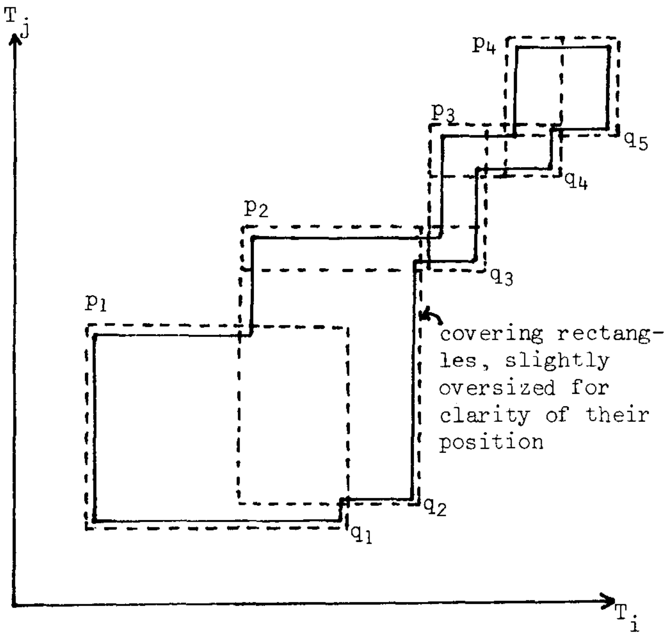

T_j
T_i
p1
p2
p3
p4
q1
q2
q3
q4
q5
covering rectangles, slightly oversized for clarity of their position

Figure 4.2: Covering the hull of Figure 4.1

The coordinates of the rectangles correspond to the lock and unlock steps to be inserted into the transactions. We choose to use an extra locking variable for each rectangle, which implies that each locking variable v occurs in exactly two transactions; we say v **acts** in the corresponding plane.

We draw the needed locking variables from an unbounded resource, namely from the set $\{v_i | i \text{ a natural number}\}$, in order of increasing i. For each pair of transactions, the locks representing rectangles used to cover the hull are selected in this order. Hence, within each transaction of a pair of transactions, the locks acting in the corresponding plane occur in order of increasing i. Therefore, no deadlock between any two transactions locked by PAL can exist.

When locking a system of more than two transactions by PAL, we start with the pair T_1, T_2 of transactions. We insert the lock and unlock operations in between the read and write actions in both transactions, T_1 and T_2, as determined by the NESW-closed cr-convex hull of the conflict points in the plane (T_1, T_2). When we continue inserting lock and unlock operations into T_1, resulting from the pair T_1, T_3 of transactions by applying the inner loop of Step 4 the next time, we face a problem. The position of the lock operations with respect to the read and write actions of T_1 is clearly determined, but not the position of the new lock operations with respect to the already present ones. Inserting new locks in an arbitrary position with respect to present locks is certainly dangerous: in this way, we might create a deadlock with such locks alone. Such a deadlock can be easily avoided by arranging these adjacent locks (with no actions in between) differently, say, according to one global order. We follow this strategy and use the order of the locking variables when we add more locks to a locked transaction system containing already some of the computed locks. Note that for unlock operations, the problem does not occur; hence, we can insert unlocks arbitrarily with respect to present locks and unlocks.

Whenever a new lock v operation has to be inserted into transaction T_i between actions T_{ir} and T_{ir+1}, the insertion takes place directly before T_{ir+1}, i.e., behind all lock and unlock operations already present between T_{ir} and T_{ir+1}. The way we use new locking variables ensures that in between T_{ir} and T_{ir+1}, all lock and unlock operations will always be ordered, for all T_{ir}, T_{ir+1}. From the construction of the NESW-closed cr-convex hull we conclude that at the moment of the insertion of a lock into a transaction, the lock is directly followed by an action conflicting with the transaction in whose plane the lock acts. Later on, only locks and unlocks can be inserted in between those two. Hence at the end of Step 4 of pre-analysis locking, the next action following a lock conflicts in the plane in which the lock acts.

Fact 4.1: Let v be a locking variable which acts in the plane (T_i, T_j), and let T_{ir} and T_{js} be the first actions succeeding lv in T_i and T_j, respectively. Then there exist conflict points $(T_{ir}, T_{jr'}), (T_{is'}, T_{js})$ in the plane (T_i, T_j).

■

To see how long algorithm PAL runs in the worst case, let us analyze the runtime requirements step by step. The computation of all direct conflict points certainly does not take longer than $O(n(d + \log n))$ steps, where n is the total number of actions in τ (recall that each variable is addressed within each transaction at most twice). This bound can be achieved, for instance, by sorting the set of all actions according to the addressed variable (the variables are thought of being ordered in some arbitrary fixed order). Then for each action, a search for the variable is performed in $O(\log n)$ time, and a scan of the neighborhood of that variable in the sorted order reveals at most $O(d)$ further occurrences, corresponding to other transactions. Graph G can be constructed in $O(d^2)$ time. The connected graphs G_l can be found in time linear in the size of G, i.e. in time $O(d^2)$. The same bound holds for finding the equivalence classes E_l^s for G_l, for all l. The indirect conflict points can be determined in time linear in the size of the graph for each edge (T_i, T_j), amounting to a total of at worst $O(d^4)$, if the partitioning of edges into equivalence classes forming simple cycles is done anew for every edge of the graphs. Hence, Step 1 can be carried out in $O(n(d + \log n) + d^4)$ steps altogether. Step 2 of the algorithm can be carried out in time linear in the number of conflict points, as we can obtain the conflict points in sorted order in Step 1, that is, in no more than $O(dn + d^3)$ computation steps. Step 3 requires no time. Step 4 can be performed in $O(dn+d^3)$ steps by similar arguments as those for Step 2. Hence, the overall worst-case runtime of our algorithm is $O(n(d+\log n)+d^4)$.

Theorem 4.2: Pre-analysis locking (PAL) is a safe lock policy, i.e., PAL transforms any transaction system τ into a safe locked transaction system $L\tau$. The transformation takes $O(n(d+\log n)+d^4)$, where n is the total number of actions and d is the number of transactions in τ.

■

5. Freedom from Deadlock of PAL

In this section, we are going to show that PAL is a deadlock free lock policy, i.e., any locked transaction system constructed by PAL is free from deadlock.

Let $\tau = \{T_1, \ldots, T_d\}$ be a transaction system and let $L\tau = \{LT_1, \ldots, LT_d\}$ be the locked transaction system constructed from τ according to the definition of PAL. Let lv denote operation lock v for locking variable v, and let uv denote unlock v. In order to show that PAL is deadlock free, let us assume the contrary for some transaction system τ, namely that $L\tau$ is not deadlock free. By construction, each pair of transactions $L\tau_{ij} = \{LT_i, LT_j\}$ is deadlock free. Therefore, if $\tau = \{T_1, T_2\}$, PAL is clearly deadlock free. Hence, assume that τ contains at least 3 transactions. Then there must exist a sequence $LT'_0, \ldots, LT'_k, k \geq 2$, of transactions in $L\tau$ and a set $\{v_0, v_1, \ldots, v_k\}$ of locking variables such that

$$\begin{array}{llll}
LT'_0: & \ldots lv_k \ldots & \ldots lv_0 \ldots & \ldots uv_k \ldots \\
LT'_1: & \ldots lv_0 \ldots & \ldots lv_1 \ldots & \ldots uv_0 \ldots \\
\vdots & \vdots & \vdots & \vdots \\
LT'_i: & \ldots lv_{i-1} \ldots & \ldots lv_i \ldots & \ldots uv_{i-1} \ldots \\
\vdots & \vdots & \vdots & \vdots \\
LT'_k: & \ldots lv_{k-1} \ldots & \ldots lv_k \ldots & \ldots uv_{k-1} \ldots
\end{array}$$

Moreover, there must exist a legal partial schedule s_D containing exactly the actions up to but not including lv_i for each transaction $T'_i, 0 \leq i \leq k$. We characterize such a deadlock situation by a **deadlock cycle** $T'_k \rightarrow T'_0 \rightarrow T'_1 \rightarrow \ldots \rightarrow T'_k$ in the following, and we call s_D the corresponding **deadlock schedule.** From the definition of PAL it follows that whenever there exists a locking variable acting in a plane, then this plane contains at least one direct conflict point.

Lemma 5.1: Assume there exists a deadlock with deadlock cycle $T'_k \rightarrow T'_0 \rightarrow T'_1 \rightarrow \ldots \rightarrow T'_k$. Then for each direct conflict point $(T'_{iw'_i}, T'_{jw_j})$ in plane (T'_i, T'_j) $(j = (i+1)\text{modulo } k), T'_{iw'_i} > lv_i$ or $T'_{jw_j} < lv_j$ holds.

Proof: Assume there exists a direct conflict point $(T'_{iw'_i}, T'_{jw_j})$, and $T'_{iw'_i} < lv_i$, and $T'_{jw_j} > lv_j$. Then from the definition of PAL and the existence of the deadlock schedule we conclude the existence of a situation in the plane (LT'_i, LT'_j) as shown in Figure 5.1. Clearly this situation is a contradiction to PAL being pairwise deadlock free.

■

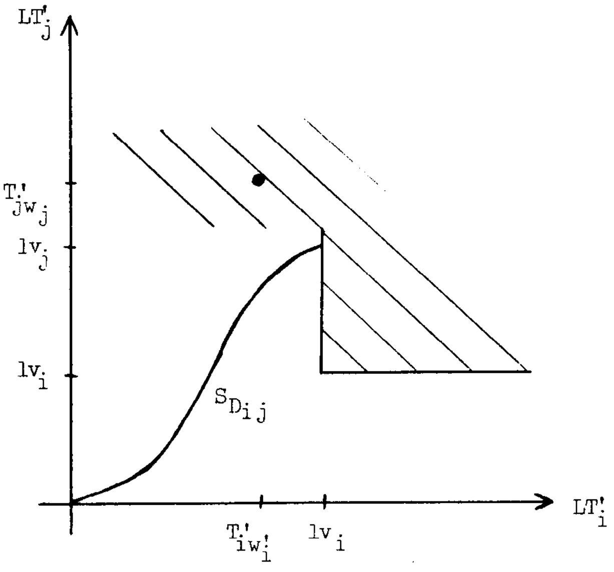


Figure 5.1: A pairwise deadlock

In the following we will derive a contradiction to $L\tau$ not being deadlock free. We will do this by systematically studying the possible locations of direct conflict points in the planes of neighbouring transactions of a deadlock cycle $T'_k \rightarrow T'_0 \rightarrow T'_1 \rightarrow \ldots \rightarrow T'_k$. We write $T'_i \xrightarrow{0} T'_j$ $(j = (i+1)$ modulo $k)$ if there is at least one direct conflict point in the plane (T'_i, T'_j) with $T'_{jw_j} < lv_j$. We write $T'_i \xrightarrow{1} T'_j$ if there is at least one direct conflict point with $T'_{iw'_i} > lv_i$. This marking of arcs is called **direct conflict point marking.** The possible direct conflict point markings of a given deadlock cycle can be characterized as follows.

Lemma 5.2: In a transaction system locked according to PAL, for each direct conflict point marking of a deadlock cycle $T'_k \to T'_0 \to T'_1 \to \ldots \to T'_k$ at least one of the following holds:

(i) there is a sequence of markings where all markings are 1.

(ii) there is a sequence of markings where all markings are 0.

(iii) there is a sequence of markings such that the markings form an alternating sequence.

(iv) there exist arcs $T'_i \xrightarrow{0} T'_j \xrightarrow{1} T'_r \xrightarrow{1} T'_s$, where $(j = (i+1)$ modulo $k)$, $r = (i+2)$ modulo $k, s = (i+3)$ modulo k,

(v) there exist arcs $T'_i \xrightarrow{0} T'_j \xrightarrow{0} T'_r \xrightarrow{1} T'_s$, where j, r, s are as in (iv).

Proof: According to PAL, there is at least one direct conflict point between any two transactions adjacent in the deadlock cycle. Therefore, every arc of every deadlock cycle possesses at least one of the markings 0,1. Assume none of (i) to (v) holds. Then there must exist at least one of the following two sequences of arcs (i, j, r, s are as in (iv)):

(1) $T'_i \xrightarrow{1} T'_j \xrightarrow{1} T'_r \xrightarrow{0} T'_s$,

(2) $T'_i \xrightarrow{1} T'_j \xrightarrow{0} T'_r \xrightarrow{0} T'_s$.

If marking 1,1,0 is part of the cycle, then start with T'_i and go backwards in the cycle until the first (next) 0 is met. This marking 0 must exist; it forms a sequence of markings 0,1,1 according to (iv) in the cycle. The case 1,0,0 is symmetric: go forward instead of backward, and exchange 1 with 0. The result then is a marking according to (v).

■

Theorem 5.3: PAL is deadlock free.

Proof: We assume that PAL is not deadlock free, and we derive a contradiction by showing that a deadlock cycle cannot have any of the forms (i) to (v) of Lemma 5.2. Therefore assume that there is a deadlock cycle of one of the following forms:

(a) a sequence of markings 0,1,1 or 0,0,1 exists in the deadlock cycle (cases (iv) or (v) of Lemma 5.2).

We show the existence of an indirect conflict point in the plane (T'_r, T'_j) according to the order of the transactions involved in the deadlock cycle starting with T'_r by constructing a conflict point path $(T'_{rw'_r}, T'_{sw_s}), \ldots, (T'_{iw'_i}, T'_{jw_j})$, where $T'_{rw'_r} > lv_r$ and $T'_{jw_j} < lv_j$. But this implies a contradiction to PAL being pairwise deadlock free, in analogy with Figure 5.1.

(b) a sequence of alternating marks exists in the deadlock cycle (case (iii) of Lemma 5.2).

W.l.o.g., let the sequence be $T'_k \xrightarrow{1} T'_0 \xrightarrow{0} T'_1 \xrightarrow{1} T'_2 \xrightarrow{0} \ldots \xrightarrow{1} T'_{k-1} \xrightarrow{0} T'_k$. From this follow direct conflict points:

$(T'_{0w'_0}, T'_{1w_1})$ and $T'_{1w_1} < lv_1$,

$(T'_{2w'_2}, T'_{3w_3})$ and $T'_{3w_3} < lv_3$,

$\vdots$

$(T'_{k-1w'_{k-1}}, T'_{kw_k})$ and $T'_{kw_k} < lv_k$.

Since in each plane of neighbouring transactions, there is at least one direct conflict point, we infer the following conflict point paths:

$(T'_{1w_1}, T'_{0w'_0}), \ldots, (T'_{3w_3}, T'_{2w'_2})$,

$(T'_{3w_3}, T'_{2w'_2}), \ldots, (T'_{5w_5}, T'_{4w'_4})$,

$\vdots$

$(T'_{kw_k}, T'_{k-1w'_{k-1}}), \ldots, (T'_{1w_1}, T'_{0w'_0})$.

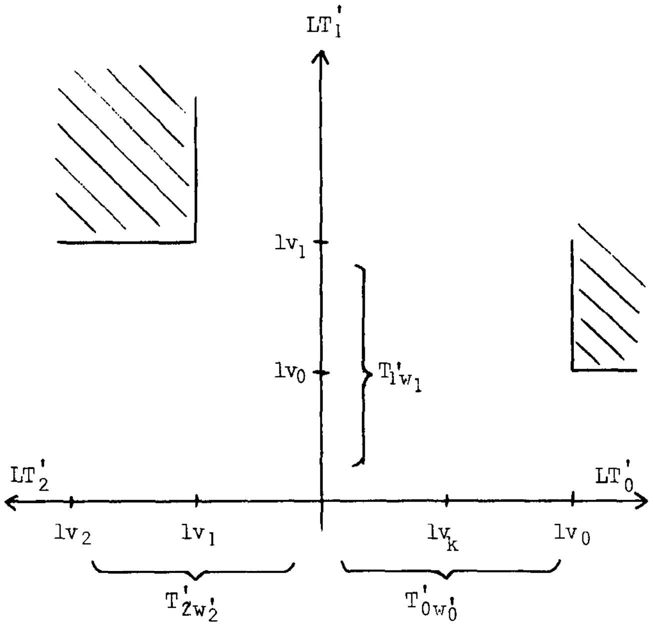


Figure 5.2: Conflicting actions in a deadlock

Since PAL is pairwise deadlock free, the following must hold (see Figure 5.2):

$T'_{2w'_2} < lv_2, T'_{4w'_4} < lv_4, \dots, T'_{0w'_0} < lv_0$. Otherwise a contradiction similar to the situation shown in Figure 5.1 follows immediately.

Hence, in each plane (T'_i, T'_j), $(j = (i+1) \text{ modulo } k)$, there exists

- a direct conflict point $(T'_{iw'_i}, T'_{jw_j})$ such that $T'_{iw'_i} < lv_i, T'_{jw_j} < lv_j$, if $T'_i \xrightarrow{0} T'_j$,
- an indirect conflict point $(T'_{iw_i}, T'_{jw'_j})$ such that $T'_{iw_i} < lv_i, T'_{jw'_j} < lv_j$, if $T'_i \xrightarrow{1} T'_j$.

Hence, since PAL constructs a connected forbidden area in each plane containing all conflict points, for any deadlock schedule s the following must hold:

$s(T'_{0w'_0}) < s(T'_{kw_k}) < s(T'_{k-1w'_{k-1}}) < \dots < s(T'_{2w'_2}) < s(T'_{1w_1}) < s(T'_{0w'_0})$, a contradiction.

(c) there is a sequence of markings that are all 0 (case (ii) of Lemma 5.2).

In each plane (T'_i, T'_j), $(j = (i+1) \text{ modulo } k)$, there exists a direct conflict point $(T'_{iw'_i}, T'_{jw_j})$ with $T'_{jw_j} < lv_j$. We can derive a contradiction analogously to case (iii).

(d) there is a sequence of markings that are all 1 (case (ii) of Lemma 5.2).

In each plane (T'_i, T'_j), $(j = (i+1) \text{ modulo } k)$, there exists a direct conflict point $(T'_{iw'_i}, T'_{jw_j})$ with $T'_{iw'_i} > lv_i$. Assume first that for at least one pair of transactions, T'_i, T'_j, $(j = (i+1) \text{ modulo } k)$, there exists some action between lv_i and lv_j in T_j. Then from Fact 4.1 it follows that there must exist some action T'_{jw_p}, namely the one closest to lv_i, with a conflict point (T'_{iw_q}, T'_{jw_p}) , where $T'_{iw_q} > lv_i$. Hence, there exists a conflict point path

$(T'_{iw_q}, T''_{1w_1}), (T''_{1w'_1}, T''_{2w_2}), \dots, (T''_{lw'_l}, T'_{jw_p}), l \geq 0$.

The following arguments are illustrated by Figure 5.3. W.l.o.g., we assume $lv_k < T'_{0w_p} < lv_0$ and $T'_{kw_q} > lv_k$.

Now consider the conflict point path

$(T'_{1w'_1}, T'_{2w_2}), (T'_{2w'_2}, T'_{3w_3}), \dots, (T'_{k-1w'_{k-1}}, T'_{kw_k})$, $(T'_{kw_q}, T''_{1w_1}), \dots, (T''_{lw'_l}, T'_{0w_p})$.

If $\{T'_0, \dots, T'_{k-1}\} \cap \{T''_1, \dots, T''_l\} = \emptyset$ then there exists an indirect conflict point $(T'_{0w_p}, T'_{1w'_1})$, which is a contradiction to PAL being pairwise deadlock free.

Now assume there exist T''_i and T'_j with $T''_i = T'_j$ and $j \neq 1$. A contradiction results from the conflict point path

$(T'_{1w'_1}, T'_{2w_2}), \dots, (T'_{j-1w'_{j-1}}, T'_{jw_j}), (T''_{iw'_i}, T''_{i+1w_{i+1}})$, $\dots, (T''_{lw'_l}, T'_{0w_p})$.

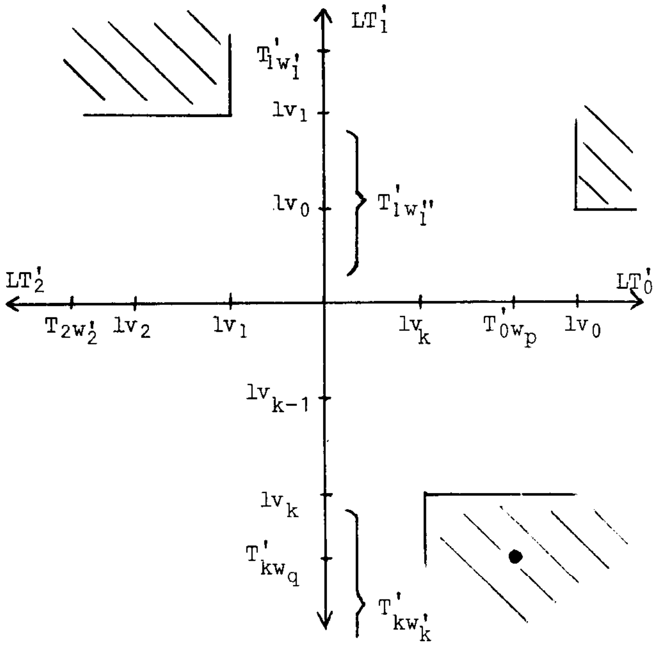


Figure 5.3: The location of conflict points in a deadlock

The case $T''_i = T'_1$ remains. In this case, there exists a conflict point path

$(T'_{1w''_1}, T''_{jw_j}), (T''_{jw'_j}, T''_{j+1w_{j+1}}), \dots, (T''_{lw'_l}, T'_{0w_p})$,

where $\{T''_j, T''_{j+1}, \dots, T''_l\} \cap \{T'_0, \dots, T'_k\} = \emptyset$. To avoid a contradiction to PAL being pairwise deadlock free, we have $T'_{1w''_1} < lv_1$. But finally consider the conflict point path
$(T'_{1w''_1}, T''_{jw_j}), \dots, (T''_{lw'_l}, T'_{0w_p}), (T'_{0w_0}, T'_{kw'_k})$,
$(T'_{kw_k}, T'_{k-1w'_{k-1}}), \dots, (T'_{3w_3}, T'_{2w'_2})$.

We conclude the existence of the indirect conflict point $(T'_{1w''_1}, T'_{2w'_2})$, which clearly contradicts PAL being pairwise deadlock free.

The remaining case is that for each pair of transactions, T'_i, T'_j, $(j = (i+1) \text{ modulo } k)$, there does not exist an action T'_{jw_p} such that $lv_i < T'_{jw_p} < lv_j$. But this immediately is a contradiction to the existence of a deadlock, since in each transaction succeeding locks without intermediate actions are ordered by PAL according to a global linear order.

■

6. Analysis and Outlook

We will analyze PAL with respect to 2PL policies. PAL and 2PL policies are conceptually different approaches. In 2PL policies, lock and unlock operations are related to entities — in PAL lock and unlock operations are only syntactic operations. Hence, not surprisingly, PAL and 2PL do not strictly dominate each other with respect to the set of allowed schedules. Figure 6.1 shows a transaction system with two schedules allowed by PAL, which cannot be allowed simultaneously by any 2PL policy.

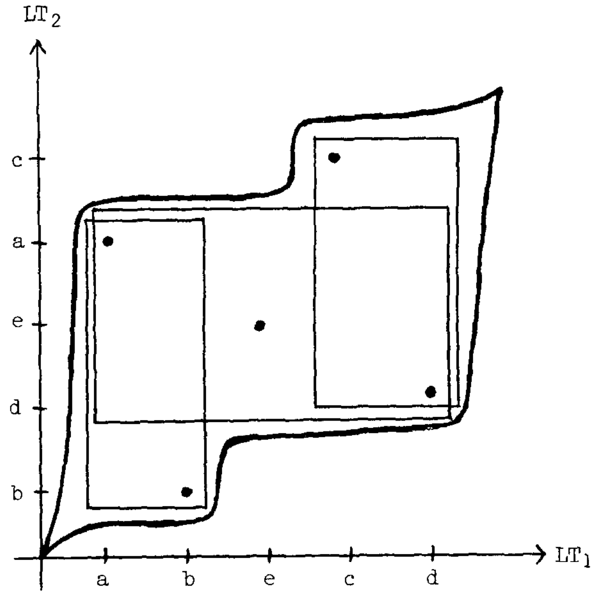


Figure 6.1: PAL is sometimes better than any 2PL policy

A simple situation in which PAL is strictly superior to preclaiming 2PL is shown in Figure 6.2. The solid lines indicate the area forbidden by PAL, the dashed lines the one forbidden by preclaiming 2PL.

A case in which 2PL policies are superior to PAL is shown in Figure 6.3. In each plane the projections of two schedules are shown: s_1 is represented by the solid lines, s_2 by the dashed lines. Assume that each transaction uses preclaiming 2PL and unlocks as early as possible. Then the solid schedule is allowed by 2PL, but is not allowed by PAL. On the other hand, the dashed schedule is not allowed by preclaiming 2PL, but it is legal in PAL.

While preclaiming 2PL can only profit from early unlocking, PAL profits from late locking and early unlocking. Our intuition therefore suggests that PAL is a deadlock free improvement over 2PL. An exact characterization of PAL versus 2PL policies is an interesting open research topic.

It is worth to note that PAL is deadlock free even though locks need not be acquired at the beginning of the transactions. To the authors' knowledge, PAL is the first safe and deadlock free general policy which is not a variant of 2PL. PAL looses potential concurrency due to its a-priori designed locking strategy; it may be improved considerably

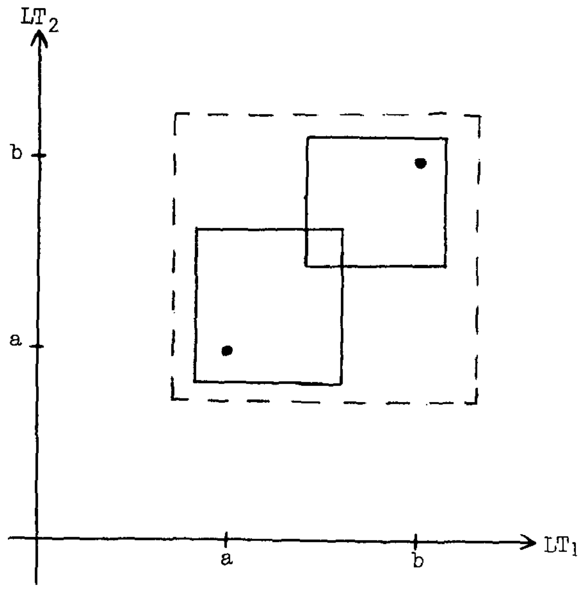


Figure 6.2: PAL is sometimes better than preclaiming 2PL

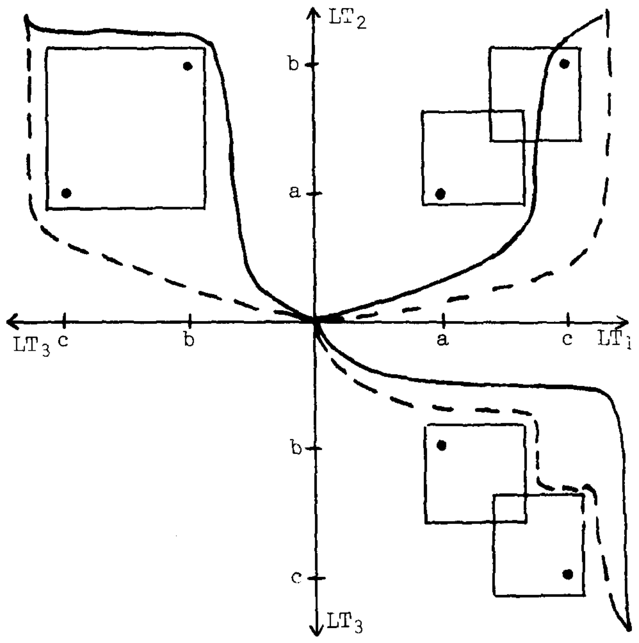


Figure 6.3: PAL and 2PL are incomparable in general

if a dynamic lock supervisor can be designed which allows to ignore lock operations (implied by the existence of indirect conflict points), if this is possible due to the prefix of a given schedule. Such a lock supervisor is the topic of future research.

Acknowledgement

The authors wish to acknowledge the TEXnical support of A. Hoth and M. Schrapp in typesetting the paper.

References

1. Eswaran, K.P., Gray, J.N., Lorie, R.A., and Traiger, I.L., "The notions of consistency and predicate locks in a database system," *Comm. Assoc. Comput. Mach.* **19** (1976), 624–633.
2. Kedem, Z., and Silberschatz, A., "Controlling concurrency using locking protocols," Proc. 20th IEEE Symp. on Foundations of Comp. Sci. (1979), 274–285.
3. Kedem, Z., and Silberschatz, A., "Non–two phase locking protocols with shared and exclusive locks," Proc. Int. Conf. Very Large Databases (1980).
4. Lausen, G., Soisalon-Soininen, E., and Widmayer, P., "Maximal concurrency by locking," Third ACM SIGACT–SIGMOD Symposium on Principles of Data Base Systems (1984), 38–44.
5. Lipski, W., and Papadimitriou, C.H., "A Fast Algorithm for testing for safety and detecting deadlocks in locked transaction systems," *Journal of Algorithms* **2** (1981), 211–226.
6. Ottmann, Th., Soisalon-Soininen, E., and Wood, D., "On the definition and computation of rectilinear convex hulls," Universität Karlsruhe, Institut für Angewandte Informatik und Formale Beschreibungsverfahren, Report 127 (1983).
7. Papadimitriou, C.H., "Serializability of concurrent database updates," *J. Assoc. Comput. Mach.* **26** (1979), 631–653.
8. Papadimitriou, C.H., "A theorem in database concurrency control," *J. Assoc. Comput. Mach.* **29** (1982), 998–1006.
9. Papadimitriou, C.H., "Concurrency control by locking," *SIAM J. Comput.* **12** (1983), 215–226.
10. Silberschatz, A., and Kedem, Z., "Consistency in hierarchical database systems," *J. Assoc. Comput. Mach.* **27** (1980), 72–80.
11. Silberschatz, A., and Kedem, Z., "A family of locking protocols for database systems that are modeled by directed graphs," *IEEE Trans. Software Eng.* **SE–8** (1982), 558–562.
12. Ullman, J.D., "Principles of database systems," Computer Science Press, Maryland (1982).
13. Wolfson, O., "A safe improvement of concurrency over two–phase–locking," manuscript, 1984.
14. Yannakakis, M., "Issues of correctness in database concurrency control by locking," Proc. 13th ACM Symp. Theory Comput. (1981), 363–367.
15. Yannakakis, M., "Freedom from deadlock of safe locking policies," *SIAM J. Comput.* **11** (1982), 391–408.
16. Yannakakis, M., "Serializability by locking," *J. Assoc. Comput. Mach.* **31** (1984), 227–244.