Reihe Informatik
3/1989

Pre-Analysis Locking: Static and Online Policies

Georg Lausen
Eljas Soisalon-Soininen

Mai 1989

Pre–Analysis Locking: Static and Online Policies [†]

Georg Lausen
Fakultät für Mathematik und Informatik
Universität Mannheim, D-6800 Mannheim, West-Germany

Eljas Soisalon-Soininen [*]
Department of Computer Science
University of Helsinki, SF-00250 Helsinki 25, Finland

Abstract

Locking is considered as a means to achieve serializable schedules of concurrent transactions. Transactions are assumed to be predeclared such that a pre–analysis for locking becomes feasible to increase concurrency. A condition for safety is introduced which, based on a pre–analysis, allows the design of policies strictly dominating known policies such as 2–phase locking. The static case, in which the complete set of transactions is known in advance, and the online case, in which a transaction is known when it is started, are considered. It is shown that a policy strictly dominating 2–phase locking and some other interesting pre–analysis policies can also be applied in an online environment.

[†] This paper contains results already presented at the International Conference on Mathematical Fundamentals of DataBase Systems, Visegrad 1989 [9], and, in a revised form, results already presented at the International Conference on Database Theory, Rome 1986 [7].

[*] The work of this author was supported by the Alexander von Humboldt Foundation.

1. Introduction

Locking as a means to achieve serializable schedules of concurrent transactions has been investigated continuously over the recent years [1,10]. Usually, locking is done according to the 2-phase locking (2PL) policy [4], which can be described by the following rules: each action of a transaction has to be surrounded by a lock/unlock pair, and all locks must precede all unlocks. This approach has two main drawbacks. First, a transaction may be forced to wait for another transaction also in cases where a non-serializable continuation cannot occur. Further, the time an entity has to be locked depends on the lock/unlock rule and not only on the position of the conflicting actions. To overcome these deficiencies a pre-analysis of the transactions becomes attractive.

A pre-analysis is proposed by Bernstein et al. [2] for a concurrency control based on time stamping to avoid unnecessary communication overhead in distributed systems. Papadimitriou [11] discusses a conflict analysis to compute so called "guardians", which are a means to guarantee that only serializable schedules may occur, i.e. which guarantee the safety of the corresponding transaction system. Yannakakis [17] suggests how to implement guardians by locking for the 2-step transaction model [11]. In a similar manner, for the multistep transaction model, we have introduced a pre-analysis to insert lock and unlock operations into transactions to achieve safety and freedom from deadlocks [6]. We call the corresponding locking policy basic pre-analysis locking (bPAL) throughout this paper. bPAL can be considered as an algorithmic continuation of the work on locking and geometry started by Yannakakis, Papadimitriou and Kung [19] and Papadimitriou [13]. In general, there is no dominance relationship between the 2PL policy and bPAL with respect to the set of accepted schedules. The intention of the pre-analysis is to maximize the potential degree of concurrency, i.e. the set of all possible schedules of the corresponding locked transaction system. It is well known, that the set of all serializable schedules of a given transaction system cannot be achieved by locking [8,12,17]. In this paper we continue our investigations on pre-analysis locking started in [6]. We will propose policies strictly dominating 2PL, and we will also discuss the case of online scheduling.

A pre-analysis to introduce early unlocking in originally 2-phase locked transactions has been proposed by Wolfson [14,15]. The resulting transaction system always allows at least as much concurrency as the original one and the method can be extended to distributed databases. However the algorithms require that the set of all transactions is given statically in advance. Wolfson further points out that locking based on pre-analysis should use locking by symbolic names, i.e.

database entities of possibly varying granularity, which are determined by examining the programs during compile time [15]. Similar remarks apply for our methods, however we use symbolic names to detect conflicting actions, for locking we use uninterpreted locks.

Uninterpreted locks and entity locks have been discussed comprehensively by Papadimitriou [10]. We use uninterpreted locks for the following reasons. Our policies include a pairwise analysis of transactions. Depending on the outcome of such an analysis lock/unlock operations are inserted into transaction pairs. To avoid side–effects between transactions in different pairs, we introduce for each pair of transactions different uninterpreted locks. Thus our policies are conceptually different from other locking policies, which, to our knowledge, all are based on entity locks. As a consequence, our policies do not belong to the hypergraph family of locking policies introduced by Yannakakis [16], which captures all known important locking policies.

In this paper we first consider the static case in which the complete set of all transactions, i.e. the transaction system, is given in advance and we introduce a new general condition that guarantees the safety of a locked transaction system. We use this condition to define improvements over 2PL. Every given locked transaction system according to 2PL can, after having been analyzed, be transformed into a safe non–2PL locked transaction system which dominates the original system with respect to the set of accepted schedules.

In order to study the problem of "online" scheduling for pre–analysis policies, we require that each transaction is completely known when started - the set of all transactions need not be known in advance. We first show that the improved 2PL policy can also be used for online scheduling. Moreover, we show that the bPAL policy can be made applicable for online scheduling. Using bPAL for online scheduling requires to compute the locks between all pairs of active transactions anew for each transaction starting its execution. To reduce this overhead we finally introduce iterative pre–analysis locking (iPAL), a policy in which locks only have to be introduced into pairs of transactions in which one of them is the new transaction.

The structure of the paper is as follows. In Section 2 we introduce the needed definitions and the geometric framework. In Section 3 we introduce our condition for safety and study static locking. Section 4 is devoted to the online case, and, finally, in Section 5, a discussion of the proposed policies concludes the paper.

2. Basic Definitions and the Geometry of Locking

A *transaction system* $\tau = \{T_1, \ldots, T_d\}$ is a set of transactions, where each *transaction* $T_i = (A_{i1}, \ldots, A_{im_i})$, $m_i \geq 1$, is a sequence of actions. Each *action* A_{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, $A_{ij} = Rx_{ij}$ meaning *read* x_{ij} and $A_{ij} = Wx_{ij}$ meaning *write* x_{ij}. We will write $R_i x$ ($W_i x$), respectively Rx (Wx), if the position of the action, respectively also the corresponding transaction, is given by the context. Two actions of two different transactions *conflict*, if they involve the same entity and at least one of them is a write action. (1)

For transaction system τ, the *(undirected) conflict graph* $\overline{D}(\tau)$ has as vertices the transactions of τ, and an edge $T_i - T_j$ whenever an action of T_i and an action of T_j conflict. A *schedule* s of τ is a permutation of all actions of τ, with the actions within each transaction in the prescribed order. If s is a schedule of τ and $T_i, T_j \in \tau$, then s_{ij} is the projection of s on the actions of transactions T_i and T_j. If A_{ij} is an action in s, then $s(A_{ij})$ denotes the position of A_{ij} in s. A schedule is *serial* if the transactions are not interleaved. For schedule s, the *(directed) conflict graph* $\overrightarrow{D}(s)$ has as vertices the transactions of τ, and an arc $T_i \rightarrow T_j$ whenever an action of T_i precedes in s a conflicting action of T_j. A schedule is called *(conflict-) serializable* if $\overrightarrow{D}(s) = \overrightarrow{D}(s')$ for some serial schedule s' of τ, or, equivalently, if $\overrightarrow{D}(s)$ is acyclic. τ is *(conflict-) safe* if every schedule of τ is serializable.

We now introduce the geometric interpretation for concurrency control [10,13,18]. Consider each transaction to be an axis in a d-dimensional coordinate system, with the actions being the coordinate values on the axes. Each pair of transactions corresponds to a plane with a grid imposed by the actions. The geometric image of a schedule in a plane (T_i, T_j) is a nondecreasing *curve* from point (0,0) to point (m_i+1, m_j+1) not passing through any other grid point. The order of the actions in the schedule is the order in which the curve intersects the corresponding grid lines. Figure 2.1 shows the curve of the schedule

$$R_1a\ W_1a\ R_2a\ R_2b\ R_1b\ W_1b\ R_1c\ W_1c\ R_2c.$$

In [6] we introduced direct and indirect conflict points to represent conflicting actions in the geometric setting.

(1) Entities may have different granularities, e.g. tuples in relations, complete relations, or sets of tuples defined by certain predicates. Thus, more precisely, we have a conflict whenever two sets have a nonempty intersection, in general. For simplicity, we will ignore this subtlety.

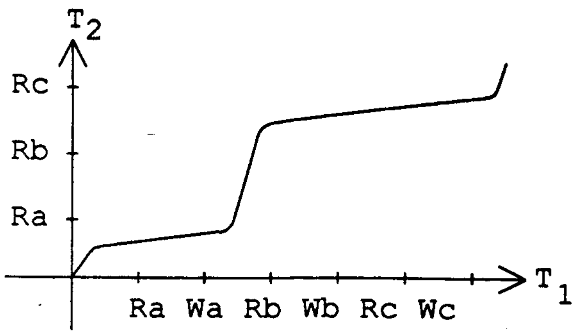

Figure 2.1 A schedule's curve.

A point $P = (A_{ip}, A_{jq})$ in plane (T_i, T_j) is a *direct conflict point* between T_i and $T_j, i \neq j$, if A_{ip} and A_{jq} conflict.

There is an edge $T_i - T_j$ in $\overline{D}(\tau)$ iff there is a direct conflict point (A_{ip}, A_{jq}) in plane (T_i, T_j). There is a path of length greater than one between T_i and T_j in $\overline{D}(\tau)$ iff there exists an indirect conflict point (A_{ip}, A_{jq}) in plane (T_i, T_j), which is defined as follows.

Point $P = (A_{ip}, A_{jq})$ is called an *indirect conflict point* between T_i and T_j, $i \neq j$, if there exists a set of transactions

$$\{T_{k_l} | 1 \leq l \leq n, n \geq 1\} \subseteq (\tau - \{T_i, T_j\}),$$

such that $(A_{k_l p_l}, A_{k_{l+1} q_{l+1}})$, for some p_l and q_{l+1}, is a direct conflict point between T_{k_l} and $T_{k_{l+1}}$ for all l, $1 \leq l \leq n-1$, and $(A_{ip}, A_{k_1 q_1})$, for some q_1, and $(A_{k_n p_n}, A_{jq})$, for some p_n, are direct conflict points between T_i and T_{k_1}, and T_{k_n} and T_j, respectively.

Direct and indirect conflict points are also called *conflict points*. Figure 2.2 shows the conflict points of a transaction system. For example, indirect conflict point $(W_1 b, R_2 d)$ is implied by the direct conflict points $(W_1 b, R_3 b), (W_3 d, R_2 d)$. If a conflict point is direct and indirect, then it is represented as direct one only.

For a two-transaction system, a schedule is conflict-serializable iff its curve doesn't separate two (direct) conflict points (cf. [10]). In a system of more than two transactions, each non-serializable schedule has a plane projection, i.e., a projection on the actions of the transactions defining the plane, such that the corresponding curve separates some conflict points. Moreover, at least one of the separated conflict points is a direct one [6]. However, some serializable schedules also separate conflict points. Figure 2.2 provides examples. The solid schedule

$$R_1 a \; W_1 a \; R_2 a \; W_2 a \; R_2 d \; W_2 d \; R_1 b \; W_1 b \; R_1 c \; W_1 c \; R_2 c \; W_2 c \; T_3$$

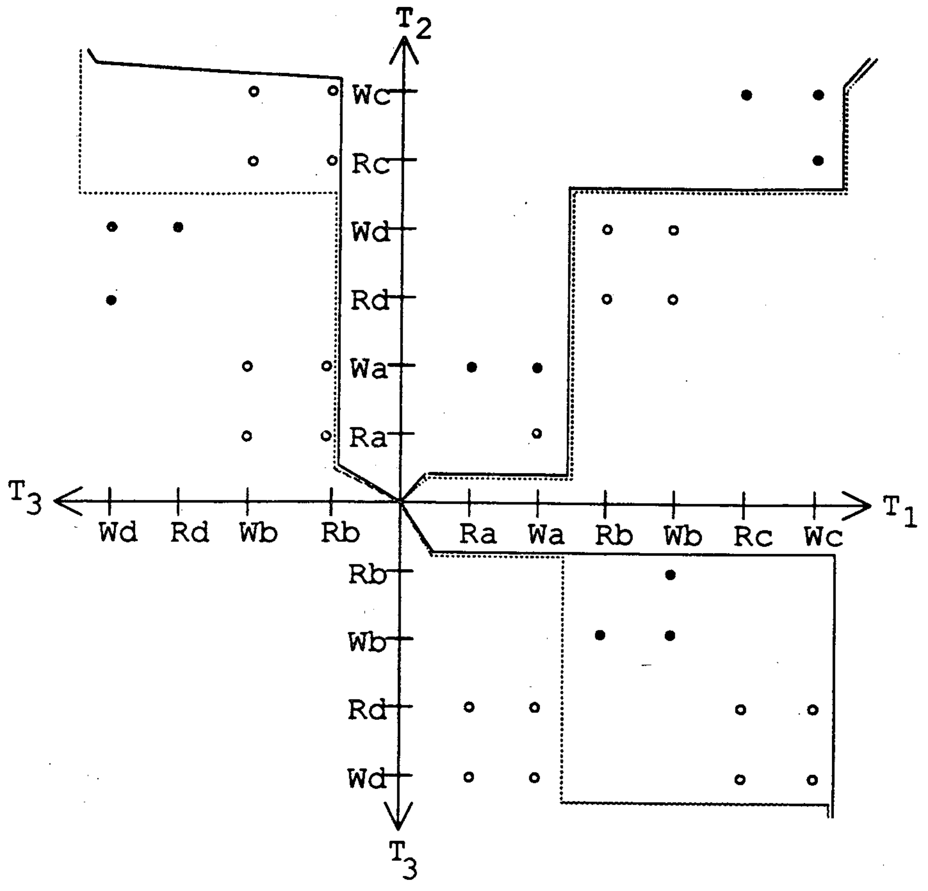

Figure 2.2 Direct (•) and indirect (∘) conflict points, and the curves of two schedules.

is serializable although conflict points are separated, the dashed schedule

$$R_1a\ W_1a\ R_2a\ W_2a\ R_2d\ W_2d\ T_3\ R_1b\ W_1b\ R_1c\ W_1c\ R_2c\ W_2c$$

is not serializable.

Let $\tau = \{T_1, \ldots, T_d\}$ be a transaction system. A *locked transaction system* $L\tau$ of τ is a set of *locked transactions*, $L\tau = \{LT_1, \ldots, LT_d\}$, where each locked transaction is a transaction that contains, besides the actions, pairs of *lock* v (Lv) and *unlock* v (Uv) operations, where v is an element of LV, the set of *locking variables*, $E \cap LV = \emptyset$, and Lv always precedes Uv. $L\tau$ can also be read as 'locked version of τ'. A locked schedule Ls of $L\tau$ is *legal*, if there is an Uv operation between any two Lv operations in Ls. The set of schedules accepted by a locked transaction system $L\tau$ is defined as $Acc(L\tau) = \{s | Ls \text{ is a legal schedule of } L\tau\}$. $L\tau$ is *safe* if $Acc(L\tau)$ contains only serializable schedules. Let $L\tau$ and $L'\tau$ be two locked versions of transaction system τ. We say $L\tau$ *dominates* $L'\tau$ if $Acc(L\tau) \supseteq Acc(L'\tau)$.

Locking variables are uninterpreted, they are not associated with entities. Instead, each locking variable v is associated with one pair of transactions, where at most one Lv and one Uv operation is in each of these transactions. More than one locking variable may be associated with the same two transactions. We use locking variables instead of the usual entity–locks. Our pre–analysis algorithms are based on a pairwise decomposition of the concurrency problem. In such a context locking variables have the appealing property that they only act in one plane, while an entity–lock in one transaction would act in all planes containing this transaction and another transaction accessing the corresponding entity.

In the geometric representation Lv and Uv operations are coordinate values on the axis of the coordinate system, too. The geometric image of a pair of Lv and Uv operations in a transaction plane is a rectangle which cannot be entered by the curve of any legal schedule [10] (cf. Figure 2.3). We call such a rectangle a *forbidden rectangle.* The region defined by the union of all forbidden rectangles in a plane is called the *forbidden region* of that plane.

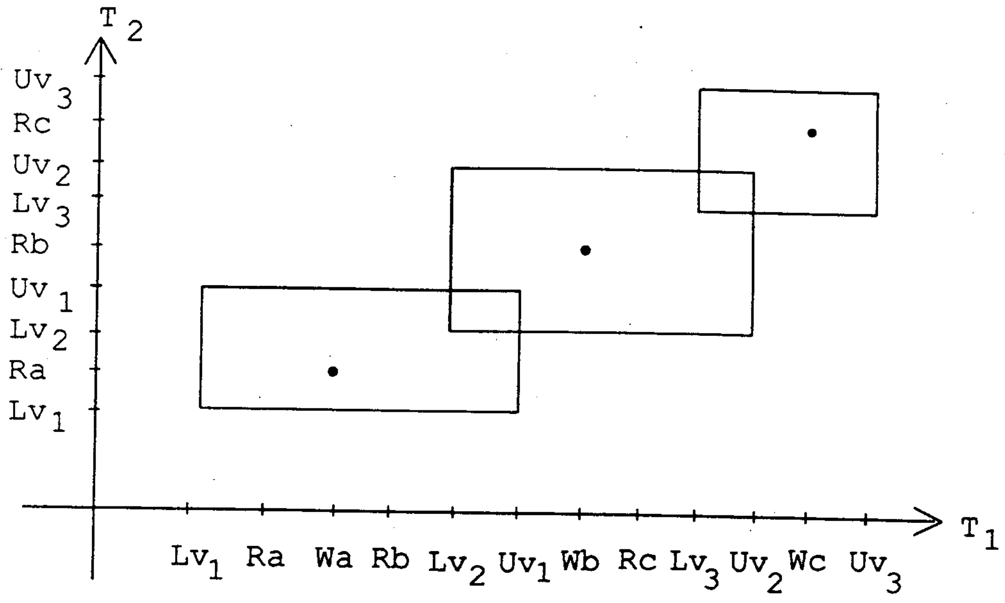

Figure 2.3 The geomerty of lock and unlock.

Forbidden regions are a geometric means to guarantee serializable schedules by enforcing an appropriate order for the actions appearing in the transactions. Let τ be a transaction system, $T_i, T_j \in \tau, i \neq j$. Let F_{ij} be a forbidden region in plane (T_i, T_j). Then for any legal schedule s of τ, if F_{ij} is connected, then F_{ij} enforces an order on the actions of transactions T_i, T_j:

$$(A_{ip_i}, A_{jq_j}), (A_{ip'_i}, A_{jq'_j}) \in F_{ij}, s(A_{ip_i}) < s(A_{jq_j}) \Rightarrow s(A_{ip'_i}) < s(A_{jq'_j}).$$

In the sequel we consider only connected forbidden regions.

The construction of forbidden regions is the critical task in the pre–analysis algorithms to achieve safe locked transaction systems. For each plane (T_i, T_j), we will be interested in a forbidden region which contains some given set $C(T_i, T_j)$ of points. Which points are selected depends on the concrete locking policy. For each plane, the construction of such a forbidden region can be done by the following steps (see [6] for details):

(1) Construct a minimal connected rectilinear region which contains the respective set of points in the plane.

(2) Compute a set of rectangles that covers the constructed rectilinear region.

(3) Make the rectangles forbidden, i.e., insert corresponding lock/unlock pairs into the transactions. The resulting locked transactions are denoted LT_i^j and LT_j^i, where LT_i^j is the locked version of T_i, and LT_j^i is the locked version of T_j.

As the forbidden rectangles define a forbidden region, we call these steps also the *realization of a forbidden region (by locking).* A corresponding locked transaction system is then derived by merging all versions LT_i^j, $1 \leq j \leq d, i \neq j$ for each transaction T_i, such that the order of the actions and lock and unlock operations for each version is preserved. The order of lock and unlock operations stemming from different versions is not necessarily unique. For example, consider $LT_i^j = \ldots A_{il} \ldots Lv \ldots A_{i(l+1)} \ldots$ and $LT_i^k = \ldots A_{il} \ldots Uw \ldots A_{i(l+1)} \ldots$, where $j \neq k$ and $v \neq w$. Thus, after merging LT_i^j and LT_i^k, Lv either precedes Uw, or Uw precedes Lv. In general, in such situations the order of lock/unlock operations may affect serializability. (Remember the lock/unlock rule of 2–phase locking.) However, for the policies we shall propose it is sufficient that for each plane (T_i, T_j) the respective set $C(T_i, T_j)$ is contained in the forbidden region. If during the merging there exists more than one possible order for lock or unlock operations stemming from different versions LT_i^j, LT_i^k, $j \neq k$, then any order may be chosen. Thus, starting with an unlocked transaction system we can derive a locked version of this system by realizing forbidden regions and afterwards merging the resulting locked versions of each transaction.

3. On Safe Static Locking Policies

A *(static) locking policy* P is a function that maps a transaction system τ to a set of locked versions of τ. If $L\tau \in P(\tau)$, we say that $L\tau$ is locked according to P. Locking policy P is called *safe*, if any $L\tau$ locked according to P is safe.

This definition of a locking policy differs from [17]. As we will analyze complete transaction systems, we define locking policies on transaction systems and not only on transactions. Later we will also define online locking policies (Section 4).

Locking policy P_1 *dominates* locking policy P_2, if for every transaction system τ there holds:

$$L\tau \in P_2(\tau) \Rightarrow \text{there exists } L'\tau \in P_1(\tau) \text{ such that } L'\tau \text{ dominates } L\tau.$$

In the sequel we will define locking policies by nondeterministic algorithms, which, for any given transaction system, can derive any locked transaction system in the corresponding image. In practice, such nondeterministic algorithms will be implemented deterministically such that from the set of possible locked transaction systems one is selected according to a fixed strategy. We therefore distinguish between a locking policy P and a concrete *implementation* of P.

Basic pre–analysis locking (bPAL) [6] is a locking policy that maps any transaction system τ to a set of locked transaction systems. A forbidden region is created for each pair of transactions T_i, T_j whose plane contains at least two conflict points, one of which is a direct one, such that all conflict points in that plane, direct and indirect ones, are included. This construction of forbidden regions guarantees that no non–serializable schedule is legal, but in general also forbids serializable schedules. In [6] we have described in detail a bPAL implementation which is free from deadlock.

The locking policy 2-phase locking (2PL) maps any transaction system τ to a set of locked transaction systems such that the following two conditions are fulfilled [4]: in each locked transaction each action is surrounded by a lock and unlock operation, and every lock operation precedes all unlock operations. Usually entity–locks are used such that each action accessing entity e is surrounded by Le and Ue. In the sequel, whenever we refer to 2PL, we will assume entity–locks. Moreover, for simplicity, only one lock mode is considered. The geometric representation of a locked transaction system according to 2PL can be characterized as follows (cf. [13]). First, for any plane T_i, T_j the forbidden region is composed of overlapping rectangles having at least one point in common. Second, for any transaction triple T_i, T_j, T_k each forbidden rectangle in (T_i, T_j) overlaps all forbidden rectangles in (T_j, T_k) on their projections on the T_j-axis. A detailed discussion of bPAL versus 2PL can be found in [6,9].

We now introduce a new sufficient geometric condition for safe locked transaction systems. Similar to 2PL safety is guaranteed by overlapping forbidden regions.

However this will not imply that in adjacent planes, i.e. planes which have one transaction in common, all forbidden rectangles have to overlap. Thus the restrictive lock/unlock rule of 2PL is weakened.

Let $L\tau$ be a locked transaction system. $L\tau$ is called *overlap-locked (OL-locked)*, if

(OL1) In each plane, in which there is at least one direct conflict point, there is a forbidden region which contains all direct conflict points.

(OL2) Let $S_1 - S_2 - \ldots - S_n - S_1$, $n > 2$ be a minimal cycle in $\overline{D}(\tau)$. [2] Then there exist coordinates $P_1, \ldots, P_m$ (not necessarily grid coordinates), where $m = k \cdot n, k \geq 1$, such that

$$(P_1, P_2) \in F_{12}, (P_2, P_3) \in F_{23}, \ldots, (P_m, P_1) \in F_{m1}.$$

$F_{12}, F_{23}, \ldots F_{m1}$ are the forbidden regions of planes (S'_1, S'_2), (S'_2, S'_3), $\ldots$, (S'_m, S'_1), respectively, $\{S'_1, \ldots, S'_m\} = \{S_1, \ldots, S_n\}$, and

$$S'_1 - \ldots - S'_m - S'_1 = k(S_1 - \ldots - S_n) - S_1. \text{ [3]}$$

The points $(P_1, P_2), \ldots, (P_m, P_1)$ are called the *overlap-points* of the corresponding cycle.

Observation 1: Any OL-locked transaction system $L\tau$ is safe.

Proof: Let s be a schedule which is not serializable, i.e., $\overrightarrow{D}(s)$ contains a minimal cycle $S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n \rightarrow S_1$. As all direct conflict points in a plane are contained in the forbidden region, we can assume $n > 2$. Let (A_{ip_i}, A_{jq_j}) be a direct conflict point of plane (T_i, T_j), where $T_i \rightarrow T_j$ is contained in the cycle. Thus $s(A_{ip_i}) < s(A_{jq_j})$. As all direct conflict points and all overlap-points in one plane are contained in the same forbidden region, we can conclude $s(P_i) < s(P_j)$ for all overlap-points (P_i, P_j) of the corresponding cycle. Since this holds for any pair of neighbouring transactions in the cycle, $s(P_1) < s(P_1)$ would be implied if $L\tau$ accepted s, a contradiction. ∎

It is worth to note that condition (OL2) requires more than mere overlapping forbidden regions of adjacent planes of a cycle. Figure 3.1 gives an example where

[2] A cycle $n_1 - \ldots - n_n - n_1$ is called minimal, if $n_i \neq n_j$ for $1 \leq i < j \leq n$, and there are no other edges in the underlying graph between any two nodes in the cycle.

[3] For $k \geq 1$ we denote by $k(n_1 - \ldots - n_n)$ the path in which $n_1 - \ldots - n_n$ is repeated k times.

adjacent planes have overlapping forbidden regions, however (OL2) is violated and the locked transaction system is not safe as the indicated schedule

$$R_1 d\ R_2 g\ W_2 a\ R_2 a\ R_3 b\ W_3 a\ W_2 g\ W_1 b\ W_1 f\ W_2 f$$

is not serializable.

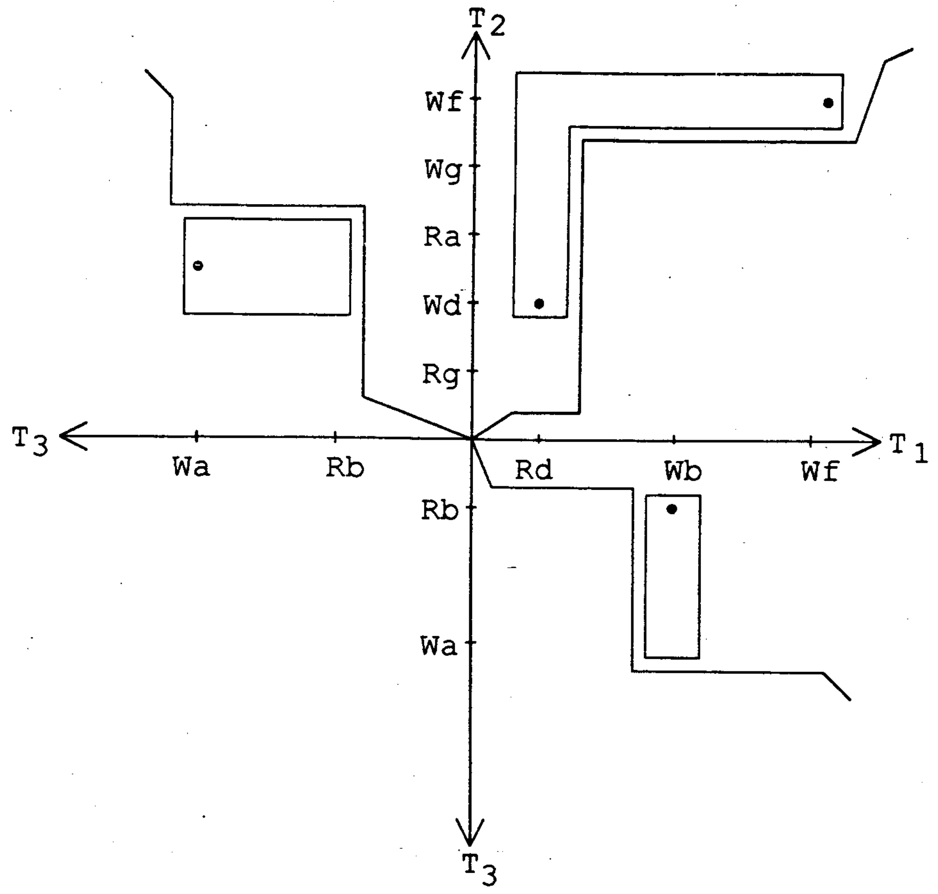

Figure 3.1 A locked transaction system which is not safe although forbidden regions in adjacent planes overlap.

OL-locking is not necessary for safety. As we are not restricted to entity-locks, safety by locking can also be achieved as shown in Figure 3.2. The locked transaction system is derived by the iterative PAL algorithm (cf. Section 5). Here, safety is achieved by locking also in planes which do not contain a direct conflict point.

It is easy to show that the 2PL policy and the bPAL policy derive OL-locked transaction systems [9].

We shall now present efficient static locking policies based on OL-locking which dominate 2PL. We assume that the transaction system τ is a priori known. The dynamic online case is treated in the next section.

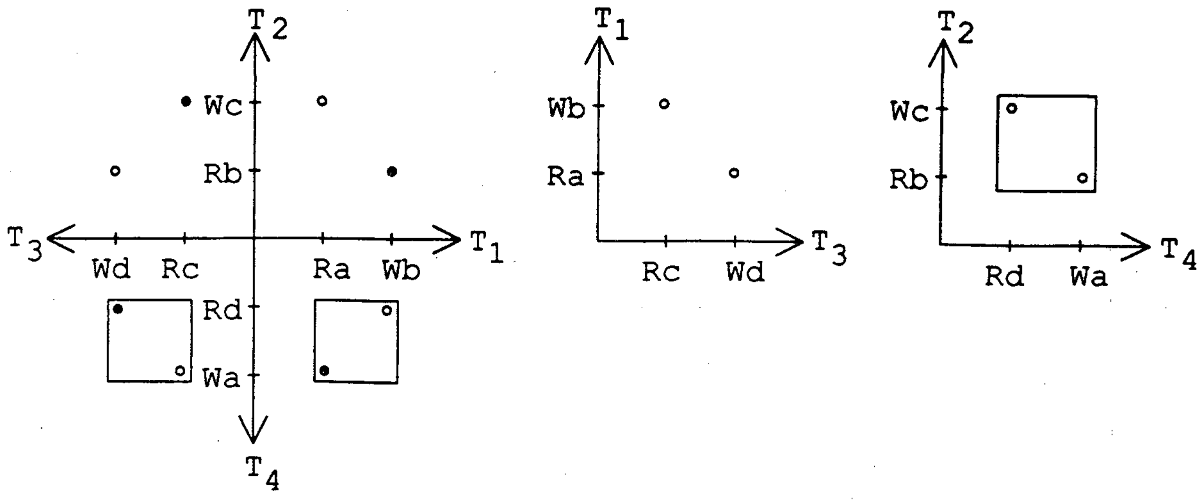

Figure 3.2 A safe locked transaction system which is not OL-locked.

The policy *overlap-point locking (OL)* selects for each transaction one unique overlap–point coordinate. For any given transaction system the corresponding set of locked transaction systems is derived by the following nondeterministic algorithm OL:

Algorithm OL

Let $\tau = \{T_1, \ldots, T_d\}, d \geq 2$, be a transaction system. The algorithm OL considers each transaction pair and may realize forbidden regions in the corresponding planes in order to construct a locked transaction system $L\tau$.

1. For each transaction T_i select one coordinate on the T_i–axis as overlap–point coordinate; denote this coordinate Q_i.
2. For each pair of transactions $T_i, T_j, i \neq j$, initialize the set $C(T_i, T_j)$ to contain the direct conflict points of plane (T_i, T_j). If $C(T_i, T_j)$ is not empty, then add overlap–point (Q_i, Q_j) to $C(T_i, T_j)$.
3. For each pair of transactions $T_i, T_j, i \neq j$ such that $C(T_i, T_j)$ is not empty, realize a forbidden region which contains all points in $C(T_i, T_j)$. The resulting locked transactions are denoted LT_i^j and LT_j^i, respectively.
4. For each transaction T_i and each pair of transactions $T_i, T_j, i \neq j$, merge the locked transactions LT_i^j to LT_i. $L\tau$ is then the set of all such $LT_i, 1 \leq i \leq d$. ■

Theorem 1: Policy OL is safe and dominates policy 2PL.

Proof: Safety follows analogously to the proof of Observation 1.

Then let $L\tau$ be any locked transaction system according to 2PL. We will show that algorithm OL can construct a locked transaction system $L'\tau$ which dominates $L\tau$. To this end choose in step 1 as coordinates of the overlap–point the first unlock operation in each locked transaction in $L\tau$. It follows that the overlap–points are contained in the forbidden regions of $L\tau$. Then, in step 3, realize forbidden regions in such a way, that each forbidden region of $L'\tau$ is contained in the corresponding forbidden region of $L\tau$. Since the forbidden regions in $L\tau$ contain all direct conflict points and the respective overlap–point, such forbidden regions always exist for $L'\tau$. ▪

Figure 3.3 shows a locked transaction system according to OL which dominates a transaction system originally locked according to 2PL.

OL is an improvement over 2PL in the sense that the forbidden regions can be made smaller. An even better policy can be obtained if we combine OL with a pre–analysis of the transaction system τ to decide for each pair of transactions T_i, T_j whether there exists a cycle of length greater than 2 in $\overline{D}(\tau)$ containing an edge $T_i - T_j$. Obviously, only in those cases an overlap–point has to be introduced in plane (T_i, T_j). We shall call the resulting policy ***overlap-point pre-analysis locking (OLPAL)***.

Algorithm OLPAL

Let $\tau = \{T_1, \ldots, T_d\}, d \geq 2$, be a transaction system. The algorithm OLPAL considers each transaction pair and may realize forbidden regions in the corresponding planes in order to construct a locked transaction system $L\tau$.

1. For each transaction T_i select one coordinate on the T_i–axis as overlap–point coordinate; denote this coordinate Q_i.
2. For each pair of transactions $T_i, T_j, i \neq j$, initialize the set $C(T_i, T_j)$ to contain the direct conflict points of plane (T_i, T_j).
3. For each pair of transactions $T_i, T_j, i \neq j$ such that $C(T_i, T_j)$ is not empty, add overlap–point (Q_i, Q_j) to $C(T_i, T_j)$ whenever there exists a path $T_i - T_k - \ldots - T_j$ in $\overline{D}(\tau)$, where $T_k \in \tau \setminus \{T_i, T_j\}$.
4. For each pair of transactions $T_i, T_j, i \neq j$ such that $C(T_i, T_j)$ contains at least two points, realize a forbidden region which contains all points in $C(T_i, T_j)$. The resulting locked transactions are denoted LT_i^j and LT_j^i, respectively.
5. For each transaction T_i and each pair of transactions $T_i, T_j, i \neq j$, merge the locked transactions LT_i^j to LT_i. $L\tau$ is then the set of all such $LT_i, 1 \leq i \leq d$.

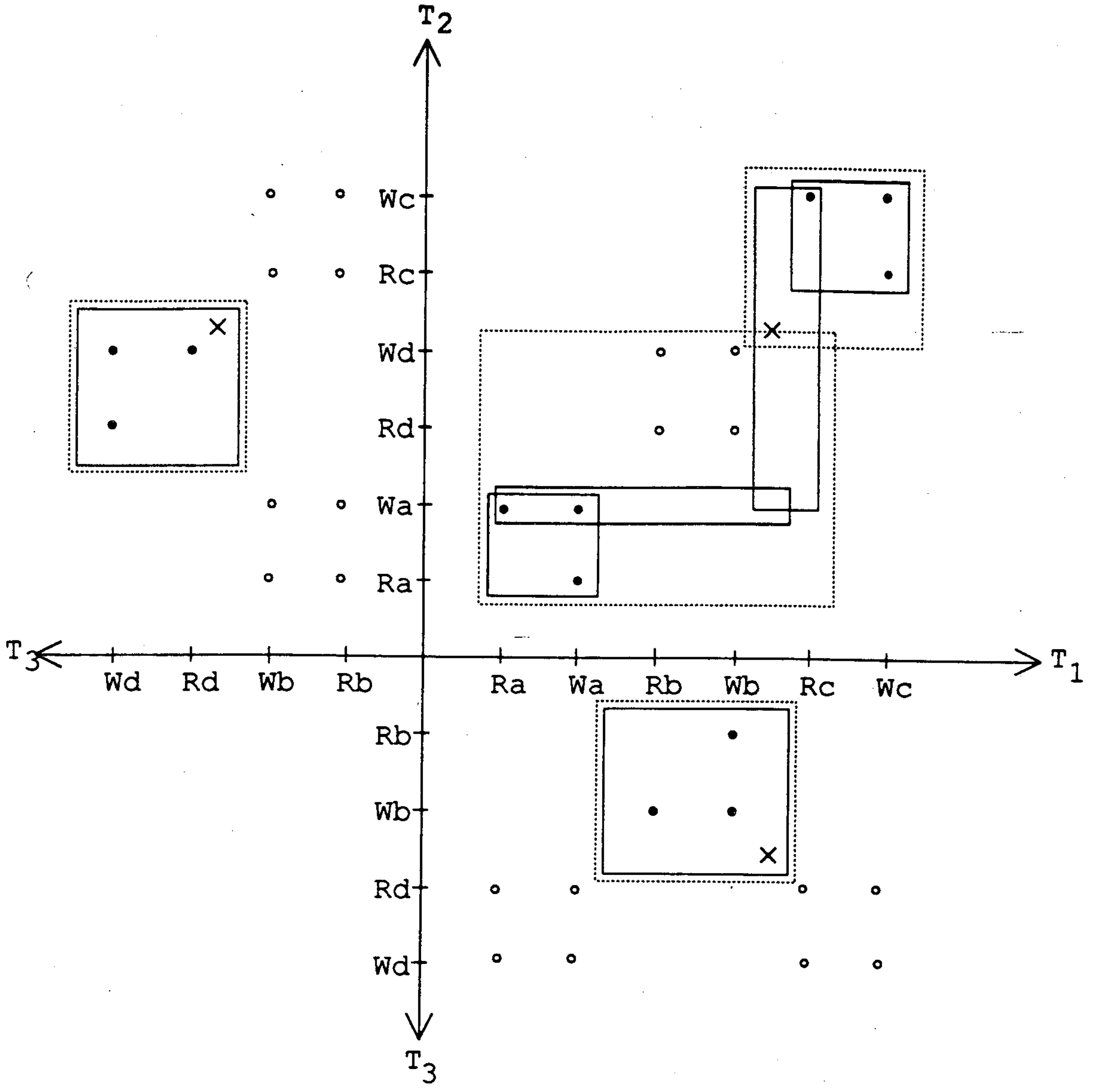

Figure 3.3 A OL-locked safe transaction system which dominates a 2PL locked transaction system (dashed lines). The overlapp-points are denoted by x.

∎

The following theorem is now straightforward:

Theorem 2: Policy OLPAL is safe and dominates policy OL. ∎

To conclude this section we shall comment on the time complexity of appropriate implementations of algorithms OL and OLPAL. The following time bounds are taken from [6]. Let us assume that the selection of overlap–points is done in constant time for each transaction. Let n be the number of all actions of transactions in τ, and d the number of transactions. All sets $C(T_i, T_j)$ of direct conflict points can be derived in time $O(n(d + \log n))$. The realization of the forbidden regions of all planes and the merging can be done in time $O(n^2 \log n)$. Therefore an implementation of OL needs time $O(n^2 \log n)$ to derive a locked transaction system in the worst case. For each pair of transactions T_i, T_j the test for a cycle of length greater than 2 can be done in time $O(e)$, where e is the number of edges in $\overline{D}(\tau)$. Thus an implementation of OLPAL needs time $O(n^2 \log n + e^2)$, in the worst case, which is identical to the time bound of bPAL.

4. On Safe Online Locking Policies

4.1 The Problem of Online Scheduling

By a *scheduler* we mean the component of a database management system which is responsible for concurrency control. We call a scheduler *online*, if the complete transaction system need not be known in advance. However, we require that whenever a new transaction is issued by a user, then the complete (action sequence of the) transaction is made known to the scheduler; the same assumption is made e.g. in [3,5].

Online scheduling becomes more difficult for uninterpreted locks than for entity locks. The reason is that a lock on an entity influences also future transactions which may access this entity, while the same cannot be achieved for uninterpreted locks as future transactions are unknown. In other words, as uninterpreted locks can only be introduced with respect to conflicts between the currently known transactions, new conflicts may arise whenever a new transaction arrives, which may require additional locks in the already running transactions. This means the schedule processed thus far is not necessarily legal after the introduction of the new locks. A straightforward solution to this problem could be to force the new transaction to wait until it is guaranteed that the current schedule is not affected. We shall concentrate on more elegant solutions in which every new transaction can start its execution immediately after having been analyzed by the scheduler. This is possible, of course, for the 2PL policy as entity locks are used. We shall show that the OL policy, which is a 2PL dominating policy, can also be applied in the online case. Moreover, also bPAL can be adopted to the online case. That is, whenever the arrival of a new transaction would cause the current schedule to be illegal, the illegality can be ignored by removing those locks which caused the illegality. This is possible because non–serializable continuations of the schedule will be avoided otherwise. As a negative result we show that OLPAL cannot be applied to online scheduling, in general. Finally, we introduce iPAL, which is a different, more efficient online policy than bPAL*.

A scheduler can be thought of as a (deterministic) implementation of a (nondeterministic) online locking policy.

Let $L\tau$ be a locked version of transaction system τ, Ls be a prefix of a legal locked schedule of $L\tau$ and let $T' \notin \tau$ be a transaction. Finally, let $\tau' = \tau \cup \{T'\}$.

An *online locking policy* P is a function that maps a triple $(L\tau, Ls, T')$ into a set of locked transaction systems $L\tau'$. Moreover, for each transaction $T \in \tau$ and each $L\tau' \in P(L\tau, Ls, T')$, the locked version $LT \in L\tau$ equals the projection of locked version $L'T \in L\tau'$ on all actions and lock/unlock operations in LT. This condition implies that the locked version of a transaction in the new locked transaction system is derived from the locked version in the old locked transaction system by inserting new lock/unlock operations, at most.

Online locking policy P_1 *dominates* online locking policy P_2, if for any triple $(L\tau, Ls, T')$ there holds:

$$L\tau' \in P_2(L\tau, Ls, T') \Rightarrow$$

$$\text{there exists } L'\tau' \in P_1(L\tau, Ls, T') \text{ such that } L'\tau' \text{ dominates } L\tau'.$$

Can we devise an online scheduler based on a pre–analysis policy, e.g. bPAL, OLPAL, or OL? That is, when scheduling according to one of these policies, can a new transaction be issued dynamically and a new locked transaction system constructed obeying this policy such that scheduling may be continued? The basic problem here is that the partial schedule processed thus far must be a legal partial schedule also when the new transaction is present. Thus, there should not occur a transaction which contradicts the partial schedule of the current locked transaction system. More precisely, let $(L\tau, Ls, T')$ be a triple as above, and let $L\tau' \in P(L\tau, Ls, T')$. T' is called *contradictory* to Ls if there exists a pair of transactions $T_i, T_j \in \tau$ such that s_{ij} intersects the forbidden region F_{ij} implied by $L\tau'$. Figure 4.1 shows an example of a contradictory transaction (for brevity we will use in the following examples only write actions).

Assume that there are first only two transactions (cf. Figure 4.1),

T_1: Wa Wb
T_2: Wa Wc.

As there is only one conflict, T_1 and T_2 can be scheduled without locking. Assume then that the first action of T_1 and the whole transaction T_2 have been performed at the point when a new transaction

T_3: Wc Wb

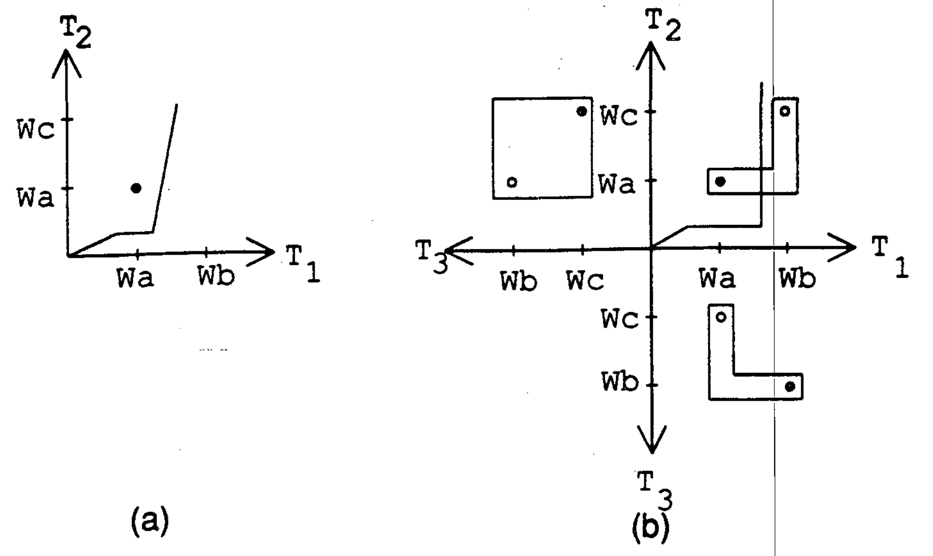

Figure 4.1 The partial schedule W_1a W_2a W_2c does not pass any forbidden region when only T_1 and T_2 are present (a) but is forbidden by bPAL policy when T_3 has arrived (b).

arrives. When bPAL is applied to $\{T_1, T_2, T_3\}$, we obtain a locked transaction system such that the already processed partial schedule

T_1: Wa

T_2: $\qquad Wa \; Wc$

is no more legal, i.e., s_{12} now intersects F_{12}, and thus we may not continue scheduling according to the new system. This example shows that contradictory transactions may occur under bPAL and therefore, bPAL cannot as such be used as an online policy.

Contradictory transactions are not an unsolvable problem for a scheduler. Whenever a new transaction arrives which turns out to be contradictory, it can be forced to wait until those transactions have finished their execution and are removed from the system which caused the contradictory situation. However, such a scheme has two main drawbacks. Firstly, there is no guarantee that the transaction can enter the system at some time point without additional mechanisms. Secondly, each time a contradictory transaction is detected the time spent to derive a new locked transaction system has been wasted. For these reasons we will call only those schedulers online which guarantee the absence of contradictory transactions.

Finally we should note that in the online case the transaction system need not grow indefinitely. A transaction T can be removed from the current system when all its actions have been performed and there is no direct conflict point (A, B), where A is an action of T, such that in the current partial schedule B appears before A. Thus, there cannot occur a cycle in the conflict graph which contains T.

There always exists a candidate for removal, since every partial schedule processed thus far is serializable.

4.2 Online Scheduling with OL

Let us first consider the online locking policy based on OL, which we shall denote OL*.

Algorithm OL*

The algorithm simulates the online situation in which new transactions may arrive while the transactions already received are being executed. Let $T_1, \ldots, T_d$ be a sequence of transactions and assume that the scheduler receives the transactions in this order. Whenever a new transaction arrives, a new locked transaction system is constructed such that for all $i = 1, \ldots, d$ the already processed schedule, denoted s^{i-1}, remains legal after the introduction of transaction T_i (s^0 is assumed to be the empty string). The current set $T_1, \ldots, T_i, 1 \leq i \leq d$, of transactions is denoted by τ^i, and A_j^{i-1} denotes the last action of T_j in s^{i-1}.

1. Let $L\tau^1 = \{T_1\}$.
2. For $i = 2$ to d perform

   2.1 For transaction T_i select one coordinate on the T_i–axis as overlap–point coordinate; denote this coordinate Q_i.

   2.2 For each pair of transactions $T_i, T_j, 1 \leq j < i$, initialize the set $C(T_i, T_j)$ to contain the direct conflict points of plane (T_i, T_j). If $C(T_i, T_j)$ is not empty, then add overlap–point (Q_i, Q_j) to $C(T_i, T_j)$.

   2.3 For each pair of transactions $T_i, T_j, 1 \leq j < i$ such that $C(T_i, T_j)$ is not empty, realize a forbidden region which contains all points in $C(T_i, T_j)$. The resulting locked transactions are denoted LT_i^j and LT_j^i, respectively.

   2.4 Let $1 \leq j < i$. For T_i, merge all LT_i^j to LT_i. For each T_j merge $LT_j \in L\tau^{i-1}$ and LT_j^i to $L'T_j$. $L\tau^i$ then is the set of all $L'T_j$, $1 \leq j < i$, and LT_i.

   2.5 For each transaction $LT_j \in L\tau^i, 1 \leq j < i$, insert those lock/unlock operations into s^{i-1} which have been introduced in step 2.3 and are to the left from A_j^{i-1}. Preserve the order with respect to LT_j. ∎

It is interesting to note that OL* does not suffer from contradictory transactions. The reason for this is that forbidden regions once computed need not to be changed when a new transaction arrives. Note, that in steps 2.3, 2.4 we have to insert lock/unlock operations also in already running transactions, respectively in step 2.5 in the current schedule. However this does not cause a problem, since new locking variables are used and the new transaction T_i has not yet started its execution. As OL dominates 2PL in the static case we have:

Theorem 3: OL* dominates online 2PL. ∎

In order to derive the time complexity of computing the new locked transaction system when the i^{th} transaction arrives, we denote by n the number of actions of all i transactions, and m_i the number of actions of the i^{th} transaction (cf. [6]). Then step 2.2 requires time $O(n \cdot m_i)$ and step 2.3 and 2.4 time $O(n \cdot m_i \log n)$. As step 2.5 can be performed in time $O(n)$, the total time complexity is $O(n \cdot m_i \log n)$. If we assume that the maximal number of actions in each transaction is bounded by a constant, we can write this time bound also as $O(i \cdot \log i)$.

4.3 Online Scheduling with bPAL

Now we consider the question of whether bPAL itself would be "safe" enough that we could ignore those forbidden regions, which imply contradictory transactions. In the above example (cf. Figure 4.1), we may ignore the forbidden region in the plane (T_1, T_2), as all non-serializable schedules of $\{T_1, T_2, T_3\}$ will be detected by the other two forbidden regions. Therefore, the solution is to build a locked transaction system that realizes only the forbidden regions of (T_2, T_3) and (T_1, T_3). According to this system the already processed schedule

T_1: Wa
T_2: $\quad\quad Wa\ Wc$

is legal and the scheduling can be continued safely.

The question is: can we indeed always forget a new forbidden region computed according to bPAL if it cuts the projected schedule processed thus far? Surprisingly, the answer is yes. We shall now outline an algorithm for constructing a locked transactions system dynamically in the sense that new transactions are allowed to arrive when transactions are being processed. The construction is done such that the schedule processed thus far is always legal with respect to the current locked transaction system.

Algorithm bPAL*

The algorithm simulates the online situation in which new transactions may arrive while the transactions already received are being executed. Let $T_1, \ldots, T_d$ be a sequence of transactions and assume that the scheduler receives the transactions in this order. Whenever a new transaction arrives, a new locked transaction system is constructed such that for all $i = 1, \ldots, d$ the already processed schedule, denoted s^{i-1}, remains legal after the introduction of transaction T_i (s^0 is assumed to be the empty string). The current set $T_1, \ldots, T_i, 1 \le i \le d$, of transactions is denoted by τ^i, and A_j^{i-1} denotes the last action of T_j in s^{i-1}.

1. Let $L\tau^1 = \{T_1\}$.
2. For $i = 2$ to d perform

   2.1 Let Ls^{i-1} be a prefix of a legal schedule of $L\tau^{i-1}$ such that Ls^{i-2} is a prefix of Ls^{i-1} [(4)].

   2.2 For each pair of transaction T_j, T_k, $1 \le j < k \le i$, determine the set $C(T_j, T_k)$ defined by:

   $C(T_j, T_k) = \{(A_{jp}, A_{kq}) \mid (A_{jp}, A_{kq})$ is a direct or indirect conflict point of τ^i in the plane (T_j, T_k) and whenever indirect and $j, k < i$, then A_{jp} is to the right from A_j^{i-1} in T_j and A_{kq} is to the right from A_k^{i-1} in $T_k\}$.

   2.3 For each pair of transactions $T_j, T_k, 1 \le j < k \le i$ such that $C(T_j, T_k)$ contains at least two conflict points one of which is a direct one, realize a forbidden region which contains all points in $C(T_j, T_k)$. To this end first take the forbidden region with respect to $L\tau^{i-1}$. If this forbidden region does not contain all points in $C(T_j, T_k)$, construct a new forbidden connected rectilinear region by enlarging the old one (by adding new rectangles) such that it contains all points in $C(T_j, T_k)$ and does not intersect the current schedule's curve. The resulting locked transactions are denoted LT_i^j and LT_j^i, respectively.

   2.4 For each transaction T_i and each pair of transactions $T_i, T_j, i \ne j$, merge the locked transactions LT_i^j to LT_i. $L\tau^i$ then is the set of all such $LT_i, 1 \le i \le d$.

   2.5 For each transaction $LT_j \in L\tau^i, 1 \le j < i$, insert those lock/unlock operations into s^{i-1} which have been introduced in step 2.3 for the new rectangles and are to the left from A_j^{i-1}. Preserve the order with respect to LT_j.

∎

(4) Here we assume that between the arrival of T_{i-1} and the arrival of T_i the schedule Ls^{i-2} is continued to Ls^{i-1}.

Similar to OL*, bPAL* does not suffer from contradictory transactions. However the reasons here are that we either consider conflict points, whose coordinate actions have not yet been executed (cf. step 2.2), or conflict points, which have as one coordinate action an action of the new transaction. Note further, that in step 2.3 we always reuse the lock and unlock operations of the previous iteration round. This guarantees that the old locked version of a transaction always can be derived by projection from the new locked version as it is required for an online policy. Step 2.3 contains a minor geometric problem. It might be the case that the curve of schedule Ls^{i-1} already passed below or above the complete forbidden region of a plane with respect to $L\tau^{i-1}$. If now the old forbidden region has to be enlarged to contain a new point (whose actions are not part of $L\tau^{i-1}$), then this task has to be performed without intersection with the schedule's curve. However, as the schedule either passed above or below the old forbidden region, we can always manage to enlarge the old region appropriately. We will not go into further details here, the precise geometric algorithms are beyond the scope of this paper. Finally, similar to OL*, in step 2.5 lock/unlock operations can be inserted in the running schedule without introducing an illegality.

The following theorem states that a locked transaction system derived by the bPAL* algorithm accepts only serializable schedules.

Theorem 4: Let $L\tau^i, 1 \leq i \leq d$, be a locked transaction system constructed by the algorithm bPAL*. Whenever Ls^i is a legal schedule of $L\tau^i$ with prefix Ls^{i-1}, schedule s^i is serializable with respect to τ^i.

Proof: The proof is by induction on i. Clearly, for $i = 1$ the claim holds. Then let $i \geq 2$, and assume that s^i is a non-serializable schedule of τ^i. We shall show that $L\tau^i$ forbids s^i after having processed the partial schedule Ls^{i-1}, a prefix of a legal schedule of $L\tau^{i-1}$. That is, we shall show that there is a projection of s^i on a transaction pair such that this projection will pass - after the arrival of the latest transaction - through two conflict points.

If s^i is a non-serializable schedule of τ^{i-1}, then, by the induction hypothesis, s^i is forbidden by $L\tau^{i-1}$. Thus the graph $\overrightarrow{D}(s^i)$ contains a minimal cycle

$$S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_n \rightarrow S_{n+1} = S_1, n \geq 1,$$

where one transaction, say S_k, is T_i, the transaction not yet included in τ^{i-1}. Assume that each arc $S_u \rightarrow S_v$ in the cycle is implied by a direct conflict between action A_{up_u} and succeeding action A_{vq_v}. If there exist more than one pair implying the respective arc, consider only one of these and call the corresponding actions the actions in the cycle. We may further assume without loss of generality that action A_{1p_1} of S_1 appears in s^i before all other actions of the cycle. There are two cases to consider depending on the form of s^i with respect to the actions in the cycle.

case 1. The actions of $S_{k+1}, \ldots, S_n$ appearing in the cycle lie in s^i after A_{kp_k} of S_k, $1 \leq k \leq n$. Then s^i is either of the form:

$$\begin{array}{lllll}
S_1: & A_{1p_1} & & & A_{1q_1} \\
\vdots & & & & \\
S_k: & & A_{kp_k} & & \\
\vdots & & & & \\
S_n: & & & A_{np_n} & A_{nq_n}
\end{array}$$

(here the mutual order of A_{1q_1} of S_1 and A_{nq_n} of S_n is irrelevant), or of the form:

$$\begin{array}{llllll}
S_1: & A_{1p_1} & & & & A_{1q_1} \\
\vdots & & & & & \\
S_k: & & A_{kp_k} & & & \\
\vdots & & & & & \\
S_n: & & & A_{nq_n} & A_{np_n} &
\end{array}$$

In both cases, after the arrival of S_k, the projection of s^i on the plane (S_1, S_n) will pass through two conflict points.

case 2. At least one action of $S_{k+1}, \ldots, S_n$ appearing in the cycle lies in s^i before A_{kp_k} of S_k, $2 \leq k \leq n$. Then let $j > k$ be the smallest index such that S_j has an action appearing in the cycle and lying in s^i before A_{kp_k} of S_k. This action must be A_{jp_j} of S_j, because if it were A_{jq_j}, then $A_{j-1p_{j-1}}$ of S_{j-1} would lie before A_{kp_k} of S_k, too. Thus s^i is of the form:

$$\begin{array}{llllll}
S_1: & A_{1p_1} & & & & \\
\vdots & & & & & \\
S_k: & & & A_{kp_k} & & \\
\vdots & & & & & \\
S_{j-1}: & & & & A_{j-1p_{j-1}} & \\
S_j: & & A_{jp_j} & & & A_{jq_j} \\
\vdots & & & & &
\end{array}$$

In this case, the projection of s^i on (S_{j-1}, S_j) will pass through two conflict points. Moreover, this happens certainly after the arrival of the transaction S_k, because the involved actions of $S_{j-1}, A_{j-1p_{j-1}}$ and $A_{j-1q_{j-1}}$, both appear only after an action of S_k. ($A_{j-1p_{j-1}}$ must appear after A_{kp_k}, otherwise $A_{j-2p_{j-2}}$ would appear before A_{kp_k}, which contradicts j being the smallest index such that S_j has an action lying before A_{kp_k}.) Thus we have shown that in all cases a desired forbidden region exists and s^i is not accepted although its prefix s^{i-1} is accepted. ∎

bPAL* basically does for each transaction T^i and transaction system τ^i the same amount of work as bPAL does in the static case [6]. Therefore, a rough estimate for the time needed, which assumes that all conflict points are computed anew each iteration round, is $O(n^2 \log n + e^2)$, where n is the number of all actions in τ^i, and e is the number of edges in $\overline{D}(\tau^i)$. We conjecture, that this upper bound is large enough to capture, in comparison to bPAL, the additional geometric overhead imposed by step 2.3. Finally, as e is $O(d^2)$, and under the assumption that the maximal number of actions in each transactions is bounded, we also can write $O(d^4)$ as time complexity for bPAL*.

Theorem 5: There is no dominance relationship between bPAL* and OL*.

Proof: This follows immediately as bPAL (OL) is a special case of bPAL* (OL*), since there does not exist a dominance relationship between bPAL and OL. ∎

Our next task is to show that safe "dynamizing" as done above in the case of bPAL* is not possible for an arbitrary OL-locked policy, e.g. OLPAL for an arbitrary selection strategy of points (Q_i, Q_j). Consider the transactions

T_1: Wa Wa' Wb

T_2: Wa Wc

T_3: Wc Wb

and assume that first only T_1 and T_2 are present and that the situation is as depicted in Figure 4.2(a) when transation T_3 arrives. Figure 4.2(b) shows a possibility for choosing forbidden regions for $\{T_1, T_2, T_3\}$ according to OLPAL, and further describes how a non-serializable schedule could occur. The overlap-points are denoted by '×'.

4.4 The Iterative Pre-Analysis Locking Policy

The obvious deficiency of bPAL* is that the set of all indirect conflict points has to be recomputed for each new transaction. The following policy, called *Iterative Pre-analysis Locking (iPAL)*, bases on a different idea. For each new transaction only those pairs have to be considered in which one transaction is the new transaction. To achieve safety, iPAL may also introduce locks between transactions which have

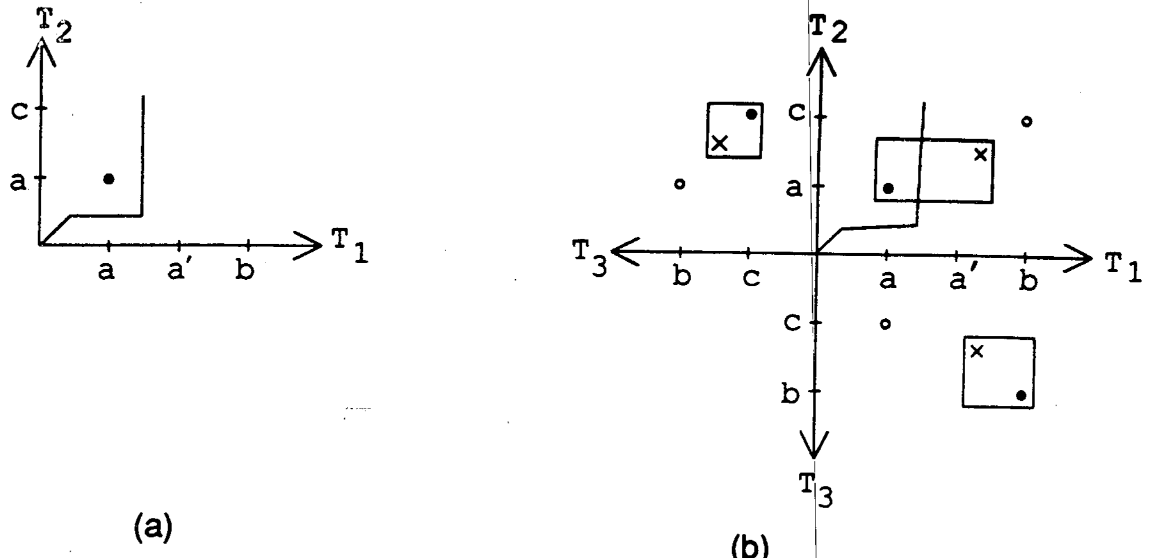

Figure 4.2 (a) The transaction system contains first only two transactions which are processed as shown. (b) The third transaction has arrived. If now forbidden region of (T_1 T_2) will be ignored, then the non-serializable schedule W_1a W_2a W_2c W_3c W_3b W_1a W_1b will be accepted.

no (directly) conflicting actions. This is necessary as iPAL may not introduce locks between transactions which have conflicting actions in situations, in which bPAL, or 2PL, for instance, would have introduced locks. iPAL therefore is a policy which behaves in a completely different way when compared with all other known locking policies.

iPAL can be used as a static, or as an online locking policy. We shall describe iPAL for the (more interesting) online case. For the static case perform the below online algorithm iPAL* and start execution of the transactions when processing of all transactions has finished.

Algorithm iPAL*

The algorithm simulates the online situation in which new transactions may arrive while the transactions already received are being executed. Let $T_1, \ldots, T_d$ be a sequence of transactions and assume that the scheduler receives the transactions in this order. Whenever a new transaction arrives, a new locked transaction system is constructed such that for all $i = 1, \ldots, d$ the already processed schedule, denoted s^{i-1}, remains legal after the introduction of transaction T_i (s^0 is assumed to be the empty string). The current set $T_1, \ldots, T_i, 1 \leq i \leq d$, of transactions is denoted by τ^i, and A_j^{i-1} denotes the last action of T_j in s^{i-1}.

1. Let $L\tau^1 = T_1$.

2. For $i = 2$ to d perform

2.1 Determine for each pair of transactions $T_j, T_i, 1 \leq j < i$ the set of direct conflict points. Denote this set $C(T_j, T_i)$.

2.2 For each pair of transactions $T_j, T_i, 1 \leq j < i$, for each sequence of direct conflict points

$$(A_{jp_j}, B_{1q'_1}), \ldots, (B_{gp'_g}, A_{iq_i}), (A_{ip_i}, C_{1q''_1}), \ldots, (C_{hp''_h}, A_{jq_j}),$$

$g + h \geq 1$, such that the corresponding transactions form a minimal cycle

$$T_j - T'_1 - \ldots - T'_g - T_i - T''_1 - \ldots - T''_h - T_j$$

in $\overline{D}(\tau^i)$, add conflict points (A_{jp_j}, A_{iq_i}) and (A_{jq_j}, A_{ip_i}) to $C(T_j, T_i)$. Moreover, whenever $p_j < q_j$ and $q_i < p_i$, then also add *extrapoint* (A_{jq_j}, A_{iq_i}) to $C(T_j, T_i)$.

2.3 For each pair of transactions $T_j, T_i, 1 \leq j < i$ such that $C(T_j, T_i)$ contains at least two conflict points, realize a forbidden region which contains all points in $C(T_j, T_i)$. The resulting locked transactions are denoted LT_j^i and LT_i^j.

2.4 Let $1 \leq j < i$. For T_i, merge all LT_i^j to LT_i. For each T_j merge $LT_j \in L\tau^{i-1}$ and LT_j^i to $L'T_j$. $L\tau^i$ then is the set of all $L'T_j$, $1 \leq j < i$, and LT_i.

2.5 For each transaction $LT_j \in L\tau^i, 1 \leq j < i$, insert those lock/unlock operations into s^{i-1} which have been introduced in step 2.3, act in plane (T_j, T_i) and are to the left from A_j^{i-1}. Preserve the order with respect to LT_j. ▮

For iPAL* the same remarks apply as for OL* with respect to contradictory transactions and the insertion of lock/unlock operations into the running schedule. Figure 3.6 shows a locked transaction system according to iPAL* when the transactions are processed in the order T_1, T_2, T_3, T_4. Observe, that forbidden regions are only introduced in planes which involve T_4. No locks are introduced in planes $(T_1, T_2), (T_2, T_3)$ even though there are two conflict points, one of which is a direct one. However, locks are introduced in plane (T_2, T_4), even though there is no direct conflict point.

Theorem 6: Let $L\tau^i, 1 \leq i \leq d$ be a locked transaction system constructed by the algorithm iPAL*. Whenever Ls^i is a legal schedule of $L\tau^i$ with prefix Ls^{i-1}, schedule s^i is serializable with respect to τ^i.

Proof: The proof is by induction on i. Clearly, for $i = 1$ the claim holds. Then let $i \geq 2$, and assume that s^i is a non-serializable schedule of τ^i. By the induction hypothesis there must exist a minimal cycle in $\overrightarrow{D}(s^i)$ which contains T_i:

$$T_j \rightarrow T_1' \rightarrow \ldots \rightarrow T_g' \rightarrow T_i \rightarrow T_1'' \rightarrow \ldots \rightarrow T_h'' \rightarrow T_j,$$

where $g + h \geq 1$. Let a corresponding sequence of direct conflict points be

$$(A_{jp_j}, B_{1q_1'}), \ldots, (B_{gp_g'}, A_{iq_i}), (A_{ip_i}, C_{1q_g''}), \ldots, (C_{hp_h''}, A_{jq_j}).$$

We then can infer indirect conflict points (A_{jq_j}, A_{ip_i}) and (A_{jp_j}, A_{iq_i}). Further, since in the planes (T_j, T_i) conflict points are contained in forbidden regions, we can follow one of the following cases:

(i) $s = \ldots\ A_{iq_i} \ldots\ A_{jp_j} \ldots$, and $s = \ldots\ A_{ip_i} \ldots\ A_{jq_j} \ldots$,

(ii) $s = \ldots\ A_{jp_j} \ldots\ A_{iq_i}\ \ldots$, and $s = \ldots\ A_{jq_j} \ldots\ A_{ip_i}\ \ldots$.

Assume case (i). As we have $T_j \rightarrow \ldots \rightarrow T_i$ and $s(A_{iq_i}) < s(A_{jp_j})$, there must exist a set of transactions $\hat{\tau} = \{\hat{T}_1, \ldots, \hat{T}_r\} \subseteq \{T_1', \ldots, T_g'\}$, such that

$$T_j \rightarrow \ldots \rightarrow \hat{T}_1 \rightarrow \ldots \rightarrow \hat{T}_r \rightarrow \ldots \rightarrow T_i$$

and, for actions $\hat{B}_{l_{pl}}$ and $\hat{B}_{l_{ql}}$ of $\hat{T}_l \in \hat{\tau}$, there holds $(1 \leq l \leq r)$:

$$s = \ldots\ \hat{B}_{1p_1} \ldots A_{jp_j} \ldots \hat{B}_{1q_1} \ldots,$$

$$s = \ldots\ \hat{B}_{lp_l} \ldots \hat{B}_{l-1p_{l-1}} \ldots \hat{B}_{lq_l} \ldots, \text{ and}$$

$$s = \ldots\ \hat{B}_{rp_r} \ldots A_{iq_i} \ldots \hat{B}_{rq_r} \ldots.$$

There exist indirect conflict points $(\hat{B}_{rp_r}, A_{iq_i})$ and $(\hat{B}_{rq_r}, A_{ip_i})$. The latter follows as we have also a path $T_i \rightarrow \ldots \rightarrow T_j$ in the cycle. We know further $s(\hat{B}_{rp_r}) < s(\hat{B}_{rq_r})$. Thus, depending on the order of A_{iq_i} and A_{ip_i}, there exist one of the situations shown in Figure 4.3. In case (a), both curves separate two indirect conflict points, which is a contradiction to our assumption $L\tau'$ being constructed by iPAL*. In case (b) the solid curve does not separate two conflict points. However, as $L\tau'$ is constructed by iPAL*, there exists also an extra point $(\hat{B}_{rq_r}, A_{iq_i})$ in the plane $(\hat{T}_r, T_i)$ (in the figure the extra point is denoted '$\times$'). Thus the solid curve separates an indirect conflict point and this extra point, which gives us a contradiction as desired.

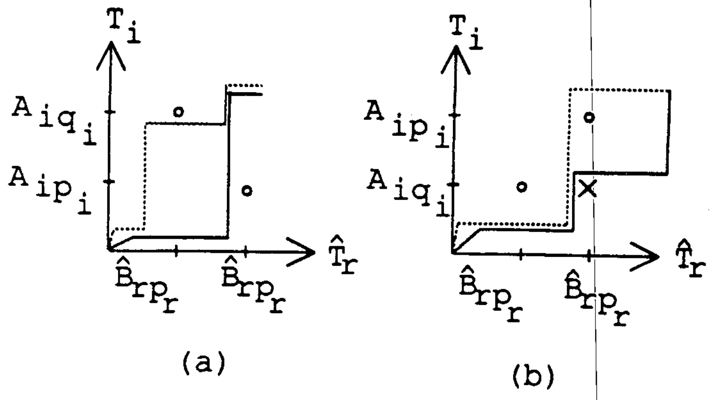

Figure 4.3 The situations in the proof of Theorem 5.

Assume case (ii). As we have $T_i \rightarrow \ldots \rightarrow T_j$ and $s(A_{jq_j}) < s(A_{ip_i})$, we can derive a contradiction by similar arguments as above. However in this case the contradiction does not depend on extra points. ∎

To see that extra points may be necessary consider as an example the transaction system $\tau = \{T_1, T_2, T_3\}$, where

$$T_1 = Wd \quad Wa \quad Wb, \quad T_2 = Wc \quad Wa, \quad T_3 = Wc \quad Wb.$$

Assume that the transactions are processed by iPAL* in the order T_1, T_2, T_3. Then the not serializable schedule

$$s = W_1d\ W_2c\ W_3c\ W_1a\ W_2a\ W_3b\ W_1b$$

will only be forbidden due to the extra point in plane (T_3, T_2) (cf. Figure 4.4, the extra point is denoted by '×'), as only in plane (T_2, T_3) a forbidden region is constructed.

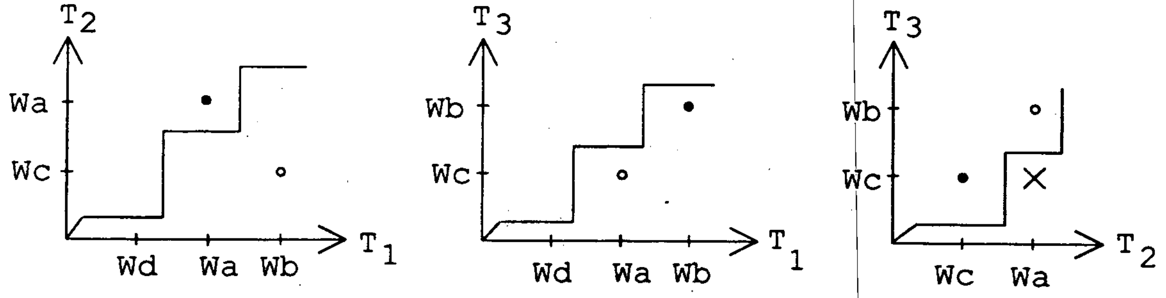

Figure 4.4 Extra points are necessary to guarantee the safety of iPAL*.

Theorem 7: There is no dominance relationship between iPAL* and bPAL*, 2PL, and OL*, respectively. ∎

The proof of this theorem follows immediately as iPAL* may introduce forbidden regions in planes which have no direct conflict point. For example, the locked transaction system in Figure 3.2 is derived by iPAL when processing the transactions in the order T_1, T_2, T_3, T_4.

Finally, we shall comment on the time complexity of iPAL*. The computation of direct and indirect conflict points can be done in a similar way as described in [6] for bPAL. However, iPAL* in each iteration step, has to consider i planes in contrast to bPAL*, which has to consider $O(i^2)$ planes. Taking this into account iPAL* needs no more than time $O(n^2 \log n + e \cdot i)$, where n is the number of all actions in τ^i and e is the number of edges in $\overline{D}(\tau^i)$. By similar arguments as above we can write this time bound also as $O(i^3)$.

5. Conclusion

In this paper we have discussed locking policies which are based on a pre–analysis of the transactions. We distinguished static policies and online policies. In the static case, the set of all transactions must be completely known in advance to perform the pre–analysis. In the online case, a transaction must be completely known when it starts its execution. Here, the pre–analysis considers the new, previously unknown transaction and the already running ones. We proposed overlap–point locking, which is applicable in the static and online case. OL* strictly dominates 2–phase locking (2PL). We therefore consider OL* as an interesting policy of practical relevance, which, for each new transaction, needs time $O(d\log d)$, where d is the number of active transactions. We further introduced two online policies, which are in the line of the pre–analysis locking policy (bPAL) introduced in [6]. Both policies have theoretically surprising properties. The time complexity for each new transaction to perform the pre–analysis is of order $O(d^4)$ for bPAL*, respectively, $O(d^3)$ for iPAL*, where d is the number of active transactions.

One characteristic property of our policies is early unlocking, i.e., locks may be relinquished before the commit point of the transactions. This has the consequence that to commit a transaction one has to wait until all predecessors of the transaction in the dependency graph have already committed. We do not consider this as a severe limitation for practical purposes, in principle. Finally, or policies are based on (syntactic) locking variables, where all known policies use entity locks (cf. [16]). Locking variables are easy to implement. For example, new variables can be introduced by cyclicly incrementing an integer counter.

6. References

[1] Bernstein, P.A., Hadzilacos, V., Goodman, N. (1987), "Concurrency control and recovery in database systems", Addison Wesley.

[2] Bernstein, P.A., Shipman, D.W., Rothnie, J.B.(1980), "Concurrency control in a system for distributed databases (SDD1)", *ACMTrans.DatabaseSystems* **5**, 18–51.

[3] Casanova, M.A., Bernstein, P.A. (1980), "General purpose schedulers for database systems," *ActaInformatica* **14**, 195–220.

[4] Eswaran, K.P., Gray, J.N., Lorie, R.A., and Traiger, I.L. (1976), "The notions of consistency and predicate locks in a database system," *Comm. Assoc. Comput. Mach.* **19**, 624–633.

[5] Katoh, N., Ibaraki, T., and Kamada, T. (1985), "Cautious transaction schedulers with admission control," *ACMTrans.DatabaseSystems* **10**, 205–229.

[6] Lausen, G., Soisalon-Soininen, E., and Widmayer, P. (1986), "Pre–analysis locking," *InformationandControl* **70**, 193–215.

[7] Lausen, G., Soisalon-Soininen, E., and Widmayer, P. (1986), "Towards online schedulers based on pre–analysis locking," Proc. Int. Conf. Database Theory, Lecture Notes in Computer Science 243, Springer Verlag, 242–259.

[8] Lausen, G., Soisalon-Soininen, E., and Widmayer, P., "On the power of safe locking," *JournalofComputerandSystemSciences*, to appear.

[9] Lausen, G., and Soisalon-Soininen, E.,"Locking Policies and Predeclared Transactions," Proc. Int. Conf. Mathematical Fundamentals of DataBase Systems, Visegrad 1989, Lecture Notes in Computer Science , Springer Verlag, to appear.

[10] Papadimitriou, C.H. (1986), "Database concurrency control", Computer Science Press.

[11] Papadimitriou, C.H. (1979), "Serializability of concurrent database updates," *J.Assoc.Comput.Mach.* **26**, 631–653.

[12] Papadimitriou, C.H. (1982), "A theorem in database concurrency control," *J.Assoc.Comput.Mach.* **29**, 998–1006.

[13] Papadimitriou, C.H. (1983), "Concurrency control by locking," *SIAM J. Comput.* **12**, 215–226.

[14] Wolfson, O. (1986), "An algorithm for early unlocking of entities in database transactions," *J.Algorithms* **7**, 146–156.

[15] Wolfson, O. (1987), "The virtues of locking by symbolic names," *J.Algorithms* **8**, 536–556.

[16] Yannakakis, M. (1982), "A theory of safe locking policies in database systems," *J.Assoc.Comput.Mach.* **29**, 718–740.

[17] Yannakakis, M. (1984), "Serializability by locking," *J.Assoc.Comput.Mach.* **31**, 227–244.

[18] Yannakakis, M., Papadimitriou, C.H., Kung, H.T. (1979), "Locking policies: Safety and freedom from deadlock," Proc. 20th IEEE Sympos. Found. of Comput. Sci., 286–297.