ON THE CORRECT SCHEDULING OF TRANSACTION SYSTEMS FOR HIGHLY PARALLEL DATA BASE MACHINES

Ravindran Krishnamurthy
Umeshwar Dayal

Department of Computer Sciences,
The University of Texas at Austin.

TR-170 Feb., 1981

ABSTRACT

This paper proposes a two-step technique for producing correct and highly parallel schedules for MIMD database machines. A parallel program schema model for transaction systems is presented. The concept of correct (i.e., serializable) executions existing in concurrency control theory for the sequential model is extended to this parallel model. The model is used to derive minimally constrained schemas for optimal scheduling. This constitutes the first step of the two-step technique. In the second step, the transactions are partially interpreted to enhance parallelism.

Table of Contents

1. INTRODUCTION	1
2. MODEL OF A TRANSACTION SYSTEM	4
2.1 Computation sequences and their properties	5
2.2 Semantic equivalence	6
2.3 Syntactic equivalence	7
2.4 Relationship between syntactic and semantic equivalence	8
3. MODEL OF EXECUTION	9
3.1 Execution and correctness criterion	9
3.2 Schema for a transaction system.	10
4. TRANSACTION MODIFICATION	13
4.1 Environment	14
4.2 Transformation of an operation pair in series	17
4.3 Generalized transformation	19
4.4 Algorithm for transformation	21
4.5 Implementation considerations	22
5. CONCLUSION	23
I. Appendix: PROOFS	26

1. INTRODUCTION

Parallel (i.e. multiple instruction-stream, multiple data-stream) data base machines such as DIRECT, have been proposed with the objective of enhancing processor utilization and achieving high transaction throughput [DeW78]. Improving processor utilization requires the efficient scheduling of transactions (for parallel execution) on available processors. But a parallel execution of transactions requiring access to shared data, can lead to race conditions and inconsistent states of the data base, unless some synchronization (concurrency control) mechanism is used[EGLT76]. This underscores the importance of both scheduling and synchronization to achieve correct (i.e. serializable) and maximally parallel executions.

The DIRECT machine uses locking in the front-end as its synchronization mechanism. However, this seems unduly restrictive and may even be prohibitive for very high throughput machines. No other synchronization mechanisms for parallel data base machines have been proposed in the literature. On the other hand, there is a wealth of concurrency control theory that has been developed for centralized and distributed data base systems, assuming a sequential model of execution [BSW79, Papa79,BSR80]. In this paper, we extend this theory to a parallel model of execution and use it to derive the minimal precedence constraints for scheduling. Although this paper develops a theoretical model of concurrency control for parallel processing environments in general, we believe that it can be directly applied to any MIMD machine such as DIRECT.

Assume that any data base system can be modelled as shown in figure 1-1. Users submit transactions (each consisting of several steps) to the system. A set of these transactions, called a <u>transaction system</u>, is input to the scheduler, which examines the transactions for potential conflicts and imposes

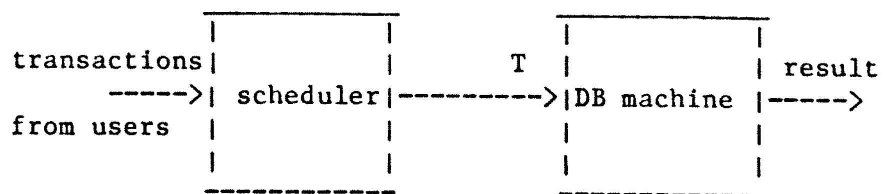


Fig. 1-1: Model of a data base management system.

a partial ordering on the transaction steps. The scheduler outputs a precedence graph (called a schema) corresponding to this partial ordering, for execution on the data base machine. The transaction system is then executed by the machine using some low-level processor allocation policy. We are concerned here with the problem of representing a transaction system using minimal precedence constraints so that any execution satisfying these constraints is correct. An execution is correct if and only if it is serializable, i.e. equivalent to some serial execution. In the existing concurrency control theory for the sequential model of execution, both in centralized and distributed systems, a transaction is modelled as a sequence of operations. The execution of a transaction system in the centralized case is also modelled as a sequence of operations, perhaps with steps of different transactions interleaved in it. This sequence is called a history. A given history is serializable if and only if it is computationally equivalent to a serial history, which defines a total ordering on the transactions in the system [BSW79, Papa79]. In the distributed case, an execution is modelled as a set of histories, one for each site. Since there is no global clock in the system, no ordering can be imposed on the operations executing at different sites. Consequently, distributed systems use protocols to ensure serializability [BSR80]. A distributed execution is serializable if and only if the history at each site is serializable and the equivalent serial histories at all sites impose the same total ordering of transactions.

A sequential ordering of transaction steps does not exploit the full power of a parallel machine. Gouda [Gou80] has extended this sequential model of a transaction and a transaction system to a directed acyclic graph (DAG). The edges of the DAG impose precedence constraints on conflicting pairs of

transaction steps. In the first part of this paper, we formalise this model using parallel program schemata theory [Kell73] and extend the notion of serializablity to this model. We then derive a minimal set of precedence constraints for a transaction system to satisfy this correctness criterion.

Most proposed concurrency control mechanisms use uninterpreted transactions, i.e. they use only syntactic information such as the read and write sets of the operations in the transaction steps. It has been shown in [KP79] that greater concurrency can be achieved if semantic information is utilized in addition to purely syntactic information. In the second part of this paper we show how to exploit semantic information to modify transaction steps and thus increase parallelism even further.

Thus, this paper proposes a two-step technique for producing correct and highly parallel schedules: first, construct a schema with minimal precedence constraints; then, modify it using semantic information to increase parallelism.

Section 2 presents a parallel program schema model of a transaction and defines the notion of syntactic and semantic equivalence of schemata. A model of execution and a correctness criterion are presented in section 3, together with a method for syntactically constructing a correct execution schedule with minimum precedence constraints. Section 4 extends the theory to incorporate semantics.

2. MODEL OF A TRANSACTION SYSTEM

In this section we adopt the general model of parallel program schemata developed in [Kell73] to data base transaction systems. The database is viewed as a shared memory M_0 and a local memory M_i for each user i. Each memory is viewed to be a countably infinite set of cells, and M is defined to be the union of all the memories. Transactions and transaction systems are modelled as schemata, defined over a set of operations. Each transaction step is an operation (henceforth, we use these two terms synonymously). Associated with each operation s_i are:

1. a unique symbol ter_i called the <u>terminator</u> of s_i; and
2. two finite sets $D_{s_i} \subseteq M$, the <u>domain</u> of s_i and $R(S_i) \subseteq M$, the <u>range</u> of s_i.

Intuitively, each operation reads the elements of its domain, performs some computation on them, and writes into the elements of its range. The terminator is an atomic event (i.e., indivisible and mutually exclusive) and represents the atomic commitment of the operation. We distinguish between read-only and other read-write operations. A read-only operation, $s_i=r_i$, reads from the main memory into the corresponding cells of the local memory, whereas a read-write operation, $s_i = w_i$, writes into the shared memory. A computation operation, $s_i=f_i$, calculates a new value from a set of values in the local memory. Informally, we shall use $R(ter_i)$ and $D(ter_i)$ are defined to mean $R(s_i)$ and $D(s_i)$ respecively. In addition for any $C \subseteq S$ define,

$$Q(C) = \{ter_i \mid s_i \in C\}, \text{ and informally } Q = Q(S),$$
$$Q = \{x \mid x \text{ is a permutation of } Q\}.$$

The domain and range of an operation provide purely syntactic information. To express the semantics of the operation (i.e. the actual computation performed by it), an <u>interpretation</u> is required. An interpretation for an

operation set defines a universe and an initial assignment of values to the memory cells, and for each operation s_i, a set of functions which map $D(s_i)$ into $R(s_i)$.

As we are interested intitially in developing a model based purely on syntactic information, a schema is defined independent of any specific interpretation. In section 4 we extend this concept to incorporate partially interpreted operations.

Definition 2-1: Let S be a set of operations. A parallel program schema , (or simply schema) over S is a directed acyclic graph G=(S,E), where the edges in E represent precedence constraints on the operations in S.

The schema specifies those operations which may be executed concurrently. An operation can be enabled for execution only after all its predecessors in G have terminated. The statement 'G is a schema over S ' will be abbreviated G $\in$ SCH(S).

2.1 Computation sequences and their properties

Based on the control information specified by a schema G $\in$ SCH(S), we identify two important properties of G, viz. equivalence and determinacy. To do this, we must characterise the set of allowable computation sequences of a given schema G.

A computation sequences (or comp) for G, is a string z $\in$ Q such that z is a topological sort sequence of the partial order defined by the schema G. Intuitively, a comp for a schema G is a sequence of terminations representing a permissible order of committing the effects of the operations. The set of computation sequences for G is denoted COMP(G).

In general, we can characterise the properties of a schema G $\in$ SCH(S) based on the properties of the set of comps allowed by G. Given an equivalence relation E on Q, G $\in$ SCH(S) is called E-determinate if

$$\forall x,y \in COMP(G) \qquad [x \equiv y \ (E)\ ]$$

that is, if COMP(G) is contained in a single E-equivalence class. Intuitively, if we define an equivalence relation E = 'produces the same result under a given interpretation', then E-determinacy ensures that all comps produce the same result under a given interpretation of the operations in the schema.

Given an equivalence relation E on Q and two schemata $G_1, G_2 \in SCH(S)$, G_1 and G_2 are said to be <u>E-equivalent</u>, (written $G_1 \equiv G_2$), iff,

$$\forall x \in COMP(G_1) \ \exists y \in COMP(G_2) \ [x \equiv y \ \ (E)] \ \ \& \ \ |\{E(x);\ x \in COMP(G_1)\}| = |\{E(y);\ y \in COMP(G_2)\}|$$

where E(x) denotes the equivalence class of x. It is to be noted here that if the schemata G_1 and G_2 are determinate, then each has only one equivalence class. Therefore, equivalence of the two schemata implies that their COMPs represent the same equivalence classes. Intuitively, both the schemata represent sets of computations that are equivalent under E.

These properties of a schema have been defined for any arbitrary equivalence relation. Two particular equivalence relations are of interest. These two relations, E_g and E_H, which represent syntactic and semantic equivalence, are defined below.

2.2 Semantic equivalence

Intuitively, we expect two schemata G_1 and G_2 to be equivalent if for any interpretation, they behave identically; i.e., given any comp x, of one schema there is a comp y, of the other schema such that for every cell m in M, x and y assign the same value to m. Since this must be true for all interpretations, we can use the notion of an Herbrand interpretation [Mann74]. This is defined formally as follows.

Definition 2-2: Given $G \in SCH(S)$, and a string $x \in COMP(G)$ define the Herbrand interpretation, (denoted $H_m(x)$), for the mth cell where $m \in M$, as follows:

1) $H_m(\lambda) = \lambda$ (where λ is the null string)
2) $\forall^m_x \in COMP(G)$, $\forall_{y<x}$, and $\forall_{ter_i \in Q}$,

$$H_m(y\,ter_i) = \begin{cases} H_m(y) & \text{if } m \notin R(s_i) \quad \langle G(y\ ter_i)\rangle \\ F_{im}(H_{1_1}(y), H_{1_2}(y), \ldots, H_{1_{|R(s_i)|}}(y)) & \text{where each } 1_i \in D(s_i) \text{ and } \\ \quad j > k \Rightarrow 1_j > 1_k \text{ . if } m \in R(s_i) & \langle G(y\ ter_i)\rangle \\ \text{Undefined otherwise} & \end{cases}$$

Where $\langle G(a)\rangle$ is true iff there exists a $b \in COMP(G)$ such that a is a prefix of b.

Intuitively, $H_m(x)$ is an encoding of the final value of the mth cell after the termination of the comp x under the Herbrand interpretaion. Given $G \in SCH(S)$, two comps $x,y \in COMP(G)$ are related by the equivalence relation E_H iff

$$\forall_{m \in M} [H_m(x) = H_m(y)] \text{ iff } x \equiv y \ (E_H)$$

. So, two comps are related by E_H iff they assign the same value to every cell for all interpretations. E_H formalizes our intuitive understanding of the semantic equivalence of two comps. It is to be noted here that an occurrence of an 'undefined' is by definition not equal to another occurrence of an 'undefined'. Thus, two schemata G_1 and G_2 are equivalent iff $G_1 \equiv G_2$ (E_H) The relation E_H correctly captures our intuitive notion of semantic equivalence, but is very difficult to detect from the definition.

2.3 Syntactic equivalence

In this subsection, we define a syntactic equivalence relation for schemata. Since this equivalence is defined in terms of the graph properties of schemata, it can be algorithmically checked. Syntactic equivalence is based on a 'reads-from' relation and a property of liveness that we define for computation sequences. These concepts were defined for histories in the models of [BSW79,Papa79]. We adapt them to our model below. Given $x = x_1x_2 \ldots x_n \in COMP(G)$, we define an augmented comp as $x_a = x_0x_1 \ldots x_nx_{n+1}$, where $D(x_{n+1}) = R(x_0) = M_0$, $R(x_{n+1}) = D(x_0) = \emptyset$. Let $Q_a = Q \cup \{x_0, x_{n+1}\}$ be the augmented

terminators. For $m \in M$, define the relation <u>reads</u> <u>m</u> <u>from</u> (denoted RF_x^m), as:

$$RF_x^m = \{(x_i,x_{n+1}),(x_0,x_i) \mid x_i \in Q\} \cup \{(x_i,x_j) \mid [(m \in D(x_j) \cap R(x_i)) \wedge (i<j) \wedge \forall_{i<k<j}[m \notin R(x_k)]]\}.$$

Define $RF_x = \bigcup_{m \in M} RF_x^m$.

Intuitively, $(x_i,x_j) \in RF_x^m$ means x_j reads the value in cell m written by x_i.

Given an augmented comp of $x \in COMP(G)$, $x_a = x_0x_1x_2 \ldots x_nx_{n+1}$, $x_i \in x_a$ is said to be <u>live</u> in x iff it satisfies the following

1. x_{n+1} is live in x; and
2. if for some live operation x_i, $(x_j,x_i) \in RF_x$ then x_j is also live in x.

Intuitively, an operation is live in a computation if its effect is evident after the computation is completed. An operation $x_i \in x$ is said to be <u>dead</u> in x iff it is not live in x.

Using the above properties, we define a syntactic equivalence relation E_g as follows. Given $G \in SCH(S)$ and $x,y \in Q$, $x \equiv y(E_g)$ iff x and y have the same live operations and the same reads-from relations for the live operations.

2.4 Relationship between syntactic and semantic equivalence

Euivalence of two comps under E_g requires that every live operation read the same set of values for all cells in its domain in both the computation sequences. From the definition of liveness, we know that the effect of only the live operations are evident in the shared memory after the computation. This observation leads to the following relationship between the two equivalence relations E_g and E_H.

Lemma 2-1: Given $x,y \in Q$ $x \equiv y$ (E_g) iff $x \equiv y$ (E_H).

This has been shown by Papadimitriou et al. [PBR77] for histories. Since comps may be thought of as possible execution histories, this result carries over to our model, and leads to the following corollaries.

Corr 2-1.1: $G \in SCH(S)$ G is E_g-determinate iff G is E_H-determinate.

Corr 2-1.2: Given two schemata, $G_1, G_2 \in SCH(S)$
$$G_1 \equiv G_2 \ (E_g) \text{ iff } G_1 \equiv G_2 \ (E_H).$$

From now on when we talk of equivalence of comps or schemata, we mean both E_g and E_H equivalence (since each implies the other).

3. MODEL OF EXECUTION

In the last section we characterized the properties of a schema. We now define some properties of an execution; in particular, we extend the concept of a correct (i.e. serializable) execution from the sequential model to our model of parallel execution. We present a sytactic procedure for deriving a schema with the property that every possible execution allowed by it is correct.

3.1 Execution and correctness criterion

Definition 3-1: An execution graph X = (Vx,Ex) of a transaction system T is a directed acyclic graph, defined by:

$$Vx = \{t_{ij} \mid i=1,2\ldots,n, \quad j=1,2,\ldots,k_i\}$$
$$Ex = \{(t_{ij},t_{kl}) \mid FIN(t_{ij}) < SRT(t_{kl})\}$$

where $FIN(T_{ij})$ = finishing time of t_{ij},
$SRT(t_{kl})$ = starting time of t_{kl}.

Intuitively, the execution graph depicts the order in which the transaction steps of T were executed. (Note that since some transaction steps were executed in parallel, this is a partial order.) To characterize the properties of X, we first define the set of computation sequences of the execution graph as follows. Given an execution graph X, its set of computation sequences, denoted COMPUT(X), is defined to be the set of topological sort sequences of X. Intuitively, COMPUT(X) is the set of sequential executions which are computationally equivalent to the parallel execution X. Informally, we can refer to x as aschema and view its edges as precedence constraints,

Consequently, we can extend the properties of schema to executions. Note that any <u>execution X permitted by a schema G</u> satisfies the precedence constraints in G. So, G must be a subgraph of X. Therefore, all properties attributable to a schema X, also hold for any schema that permits the execution X.

Definition 3-2: Given a transaction system T, and a permutation p of {1,2,...n}, (where p is viewed as a function), a <u>serial execution</u> corresponding to p, (denoted SX_p), is an execution in which all transaction steps of $T_{p(j)}$ are executed before any transaction step of $T_{p(k)}$ is executed iff $p(j) < p(k)$.

Thus, a serial execution imposes a total ordering on the transactions, and its graph is a chain.

Proposition 3-1: $COMPUT(SX_p)$ has exactly one element.

Informally, we shall denote this computation sequence also by SX_p. With the above definition of serial execution, we are ready to define the correctness criterion of serializability.

Definition 3-3: Given a computation sequence $x \in COMPUT(X)$, we say that x is <u>serializable</u> iff there exists a permutation p such that $x \equiv SX_p$. An execution is serializable iff every element of COMPUT(X) is serializable.

Proposition 3-2: X is serializable iff $X \equiv SX_p$ for some permutation p.

Thus, a parallel execution is correct iff all of its computation sequences are equivalent to some serial execution. Note that in this parallel model of execution, serializability ensures that if there are m transaction steps executing in parallel, then every one of the m! possible sequences of committment of these transaction steps is serializable.

3.2 Schema for a transaction system.

We give below a syntactic procedure to obtain a schema with the property that every resulting execution is serializable. The following definitions are adapted from [BSW79,Papa79] to our model.

Definition 3-4: Given an augmented comp for an $x \in COMP(G)$, define the <u>interferes relation</u> (denoted I_x) as:

$$I_x = \{(x_k, x_i) \mid (x_i, x_j) \in RF_x \wedge \exists_{m \in M}[(m \in D(x_j) \cap R(x_i)) \wedge \exists_{k<i}[m \in R(x_k)]]\}$$
$$\cup \{(x_j, x_k) \mid (x_i, x_j) \in RF_x \wedge \exists_{m \in M}[(m \in D(x_j) \cap R(x_i)) \wedge \exists_{k>j}[m \in R(x_k)]]\}$$

Intuitively, I_x guarantees that if x_j reads a value in cell m from x_i and x_k writes into cell m then x_k should either precede x_i or follow x_j. Using the interferes and reads-from relations, the serialization graph of a comp is defined as follows.

Definition 3-5: Given $G(V,E) \in SCH(S)$, and $x \in COMP(G)$, we define a serialization graph of x (abbreviated SR-graph of x), $G_x=(V_x,E_x)$ from a comp $x \in COMP(G)$ as follows

$$V_x = V, \ E_x = RF_x \cup I_x.$$

The edges of the SR-graph of x represent all the reads-from relations and the interferes relations for x. So if we use G_x as the schema then any comp $y \in COMP(G_x)$ must reflect this ordering. This is stated in the following lemmas (the proofs of which are given in the Appendix).

Lemma 3-1: Given $x \in COMP(G)$, $y \in COMP(G_x)$, if x has no dead operations, then y has no dead operations.

Lemma 3-2: Given $x \in COMP(G)$, $y \in COMP(G_x)$, it follows that $RF_x = RF_y$.

Lemma 3-3: Given $G \in SCH(S)$, $x \in COMP(G)$, such that there is no dead operation in x then $x \equiv y$ (E_g) for all $y \in COMP(G_x)$.

This follows directly from Lemmas 3-1 and 3-2.

So Lemma 3-3 shows that if we can choose a computation sequence x based on some correctness criterion and derive G_x, then every comp $y \in COMP(G_x)$ is also equivalent to x. The following corollary is an obvious extension.

Corr 3-3.1: G_x is E_g-determinate.

Given a serial execution SX_p, we derive a SR-graph after deleting all the dead transaction steps in SX_p. For convenience, we denote this G_p, (instead of G_{SX_p}, the notation used earlier). We would like to use G_p as the schema. For this, we show that any execution permitted by it is serializable.

Theorem 3-1: Given a SR-schema G_p of a serial execution SX_p, every execution permited by G_p is serializable.

Proof: From lemma 3-3 it follows directly that every computation sequence in $COMP(G_p)$ is serializable.
We know that every edge (i.e. precedence constraint) in G_p is also an edge in any execution X resulting from G_p. Therefore every topological sort sequence

of X must also be a topological sort sequence of G_p. So, COMPUT(X) $\subseteq$ COMP(G_p) Therefore we can conclude that every topological sort sequence of COMPUT(X) is serializable to SX_p. [QED]

Thus we have proved that G_p is a schema which represents the transaction system T, and provides sufficient information to result in a correct execution. From now on we refer to G_p as the SR-schema. To show that this schema imposes a minimal set of precedence constraints on the transaction steps in T, we prove the following theorem.

Theorem 3-2: Given a SR-schema G_p, let G_{pmin} be the schema with fewest edges such that $(G_p)^+ = (G_{pmin})^+$. (where G^+ is the irreflexive transitive closure of G). Then G_{pmin} has the necessary and sufficient precedence constraints required for any schema to be equivalent to schema SX_p.

Proof: Sufficiency follows directly from Theorem 3-1.
Necessity is proved as follows. First, observe that G_{pmin} is unique for a given G_p, as G_p is a directed acyclic graph. Now suppose we remove an edge $e=(x_i,x_y)$ from G_{pmin}.
Case 1: If e is a reads-from edge, then G_{pmin} is not equivalent to G_p $(=SX_p)$ under the equivalence relation E_g; thus it is not equivalent under relation E_H too.
Case 2: If e is an interferes edge, then (without loss of generality) let x_i read a value in cell m from x_k, and x_j write into cell m. If e is removed, then there is a computation sequence y in which x_j follows x_k, and precedes x_i. So in y, x_i reads cell m from x_j instead of from x_k. Thus y is not equivalent to SX_p, which implies that G_p is not serializable.
This proves that G_{pmin} represents the necessary and sufficient set of precedence constraints for any schema to be equivalent to SX_p. [QED]

But it is clear that this minimality is for a particular choice of serial execution SX_p, i.e. for a particular permutation p. But there are n! SR-schemata to choose from as there are that many serial permutations p. If we want to choose an optimal schema from the n! possible schemata, then minimising the number of edges may not necessarily be a good measure. In [KD81] we present five metrics for optimization, which have been culled from the literature. We show that one of these metrics is useless for this problem, and, for the remaining four, the optimization problems are intractable (i.e. NP-hard). Hence, it seems likely that finding an optimal

schema, using only syntactic information, is intractable. But Kung and Papadimitriou [KP79] have shown that semantic information can be used to achieve greater parallelism. We explore this idea in the next section.

4. TRANSACTION MODIFICATION

It was observed that the scheduling problem using syntactic information alone (i.e readsets and writesets) is intractable for all interesting performance measures. There are two ways to cope with this intractablity. One way is to find an approximation algorithm that finds a suboptimal solution. The other way is to redefine the problem, by changing the domain of optimization or by relaxing the optimality measure, so as to simplify the problem. This section shows how semantic information can be used to redefine the problem to make it simpler. One approach is to construct a schema that is serializable and then to optimize it using semantic information. Semantic information is incorporated by partially interpreting the operations. Thus, each transaction step is assumed to be a statement (retrieval or update request) in a high-level query language. We develop a set of rules for transaction modification. Each modification transforms a schema (for a given transaction system) into an equivalent schema which is better according to some performance measure. Initially this measure is assumed to be the diameter of the schema. (We point out in [KD81] the practical significance of this measure and its superiority over the others.) Later, this is extended to a measure based on data base cost.

4.1 Environment

The data base is assumed to consist of a single relation R. In a data base with more than one relation, R can be thought of as the product of all the relations. Each tuple of R has a unique identifier (TID) and corresponds to a cell of the shared memory M_0. Transaction steps are assumed to be statements written in some high-level relational calculus-based language, such as QUELO [HSW75]. As the exact syntax is not of concern here, we represent each type of statement in the following manner:

1. MODIFY: $MOD(targ_m, q_m, R)$
2. INSERT: $INS(targ_i, q_i, R)$
3. DELETE: $DEL(q_d, R)$
4. RETRIEVE: $RET(targ_r, q_r, R)$

In each statement the qualification q is a predicate that selects a subset of the relation R to be the operand of the operation; the target list targ defines the computations to be performed on the operand. In QUELO, the qualification is a boolean combination of clauses of the form <term><op><term>, where <term> is an attribute, a constant, or an arithmetic function (e.g +.*) of other terms; and <op> is an arithmetic comparison operator (e.g. $=, \leq$). The target list is a list of (attribute,term) pairs. Initially, we assume that terms in a target list are constants, but we show section 4-5 that this assumption can be relaxed. Note that we do not permit aggregation functions (e.g. average,sum) in our statements. Without loss of generality, we assume that a target list specifies every attribute in R. This assumption and our earlier assumption that the data base contains only one relation do not restrict the applicability of the theory; they are made merely to simplify the formalism.

Let Q_x be the set of tuples selected by qualification q_x and, $TARG_x$ be a function corresponding to target list $targ_x$, such that, $TARG_x(Q_x)$ is the set of tuples generated by evaluating $targ_x$ on the selected tuples Q_x. For a single tuple t, $TARG_x(t)$ is defined in the obvious manner. Figure 4-1 tabulates the effect of each operation.

We shall consider a general class of query processing strategies in which each operation is performed in two steps (as shown in figure 4-2): a qualification step q that selects the tuple ids (TIDs) of the R-tuples satisfying the qualification q; then, an effect step e that performs the operation specified by the target list on the tuples whose TIDs were selected by q. For this class of query processing strategies, the time taken to evaluate the qualification (step q) is likely to be much greater than the time taken for step e. Also, step q does not update the database. These two observations make it a prime candidate for execution in parallel with other qualification steps (appropriately modified, as described in the sequel).

Consider an ordered pair of operations with the precedence constraints shown in figure 4-3. Suppose this pair can be modified to an equivalent schema (shown in figure 4-4) such that Te_1, Te_2, Tq_1, and Tq_2 can be syntactically determined. Then, the modified operation pair has smaller diameter and so is better according to our measure of parallelism.

In developing the modifications, we use a canonical representation of the update operations (see figure 4-5), called ⧗†graph. In this representation, qd and qi are the qualifications that select the tuples to be deleted and inserted respectively; e_d and e_i are the corresponding effects. It is obvious that the modify, insert and delete operations can be modelled by choosing appropriate q's. To model retrieve operations we observe that all retrieve

In the following table R, R′, R″ are the relations before the execution, after the execution and the result of the retrieve operation. Also let

$Q_x = \{t \mid t \in R \text{ and } t \text{ satisfies } q_x\}$
and $TARG_x$ is the function corresponding to $targ_x$.

The following table lists the effect of each operation:

$MOD(targ_m, q_m, R)$	$R' = (R - Q_m) \cup TARG_m(Q_m)$	$R'' = \emptyset$
$INS(targ_i, q_i, R)$	$R' = R \cup TARG_i(Q_i)$	$R'' = \emptyset$
$DEL(q_d, R)$	$R' = R - Q_d$	$R'' = \emptyset$
$RET(targ_r, q_r, R)$	$R' = R$	$R'' = Q_r$

R_{OS} = the resulting relation after an execution permitted by OS.

R_{TS} = the resulting relation after an execution permitted by TS.

Figure 4-1: Effects of the operations.

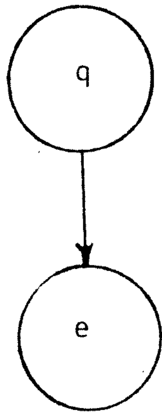


Fig. 4-2: Model of an operation

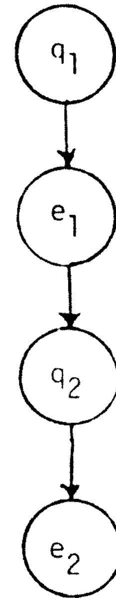


Fig. 4-3: Schema for an ordered pair of operations

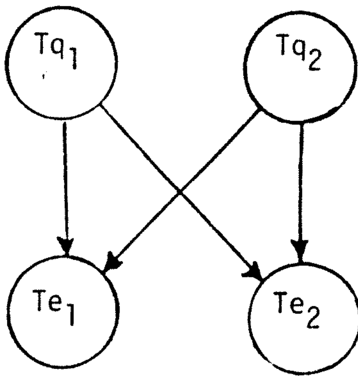


Fig. 4-4: Modified schema for an ordered pair of operations

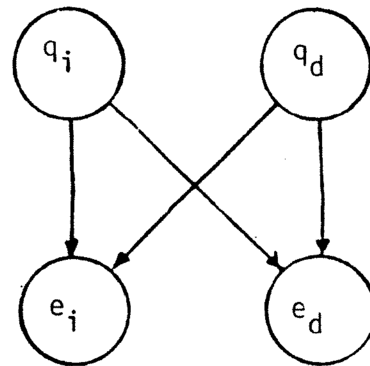


Fig. 4-5: Canonical representations of an operation graph.

operations in a transaction are live, i.e., the retrieved values are used in a subsequent update (modify,insert, or delete) operation. Hence, we can combine the retrieve operation with the update operation and correspondingly change the effect step e of the update operation. An example of such a modification is shown in Figure 4-6. From a practical standpoint, we do not expect to see dead transaction steps. At the very least, a transaction step should record the fact that it ran by writing on an output device which in our model is part of the shared memory.

An assumption that we make here is that every insert (ei) create a new tuple of relation R. Thus, every new tuple that is inserted into R is assigned a new TID. This implies that a modify operation is modelled as the deletion of old tuples and the insertion of new (modified) tuples with new TID's. This assumption in conjunction with an earlier assumption (that only constants are permitted as terms in target lists) ensures the sjointness property. It is this disjointness property that we need in our proofs fo this subsection. We argue in section 4-5 that and implementation can, in fact relax these assumptions, and still satisfy the disjointness property.

With this canonical model of an operation, we define an elementary schema called an <u>operation pair</u> as shown in figure 4-7. Note here that the I+graph may have more than one statement associated with it. For this elementary schema, we derive an equivalent schema with greater parallelism (i.e smaller diameter). Then we define a generalized canonical form of elementary schema and show how to extend the transformation to this generalized form. Further, we show that the transformed schema is of the same generalized canonical form so that it can be subsequently paired with some other operation. This closure property is useful in repetitive applications of the transformation.

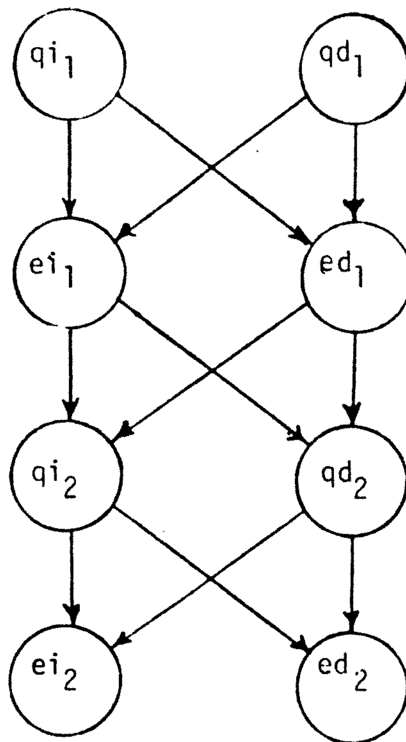


Fig. 4-7: Schema (OS) for an operation pair.

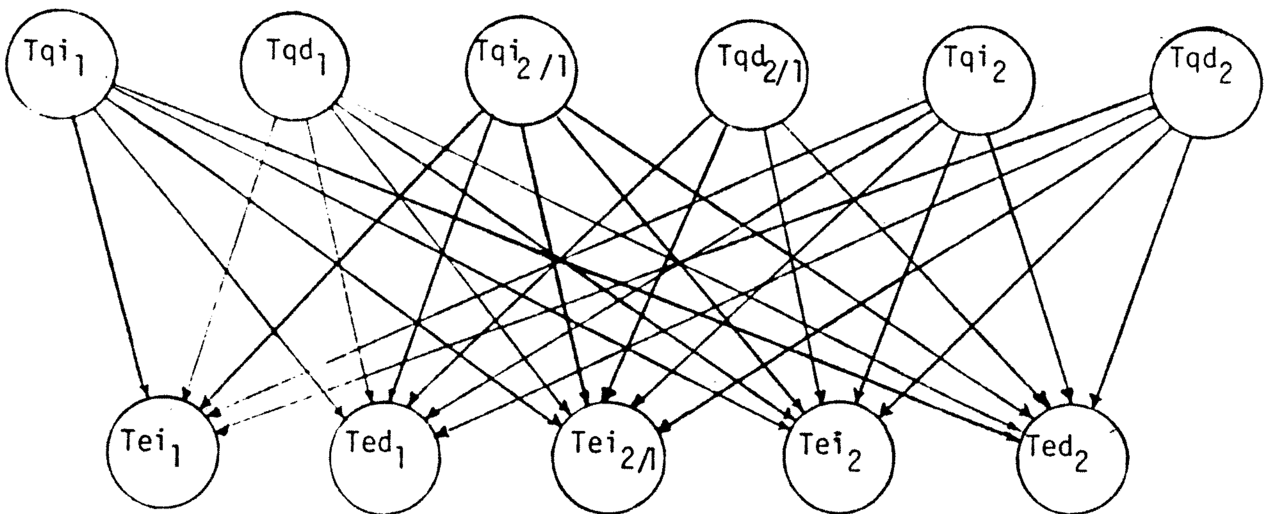


Fig. 4-8: Transformed schema (TS) for an operation pair.

Tuples selected tuples by q's in the original schema:

$OQi_1 = \{t \mid t \in R \text{ and } t \text{ satisfies } qi_1\}$
$OQd_1 = \{t \mid t \in R \text{ and } t \text{ satisfies } qd_1\}$
$OQi_2 = \{t \mid t \in (R - OQd_1) \cup OQi_1 \text{ and } t \text{ satisfies } qi_1\}$
$OQd_2 = \{t \mid t \in (R - OQd_1) \cup OQi_1 \text{ and } t \text{ satisfies } qd_1\}$

Read/write sets for e's in the original schema

$OEi_1 = OQi_1$ $\quad OEi_2 = OQi_2$
$OEd_1 = OQd_1$ $\quad OEd_2 = OQd_2$

Tuples selected by Tq's in the transformed schema:

$TQi_k = \{t \mid t \in R \text{ and } t \text{ satisfies } qi_k\} \quad k=1,2$
$TQd_k = \{t \mid t \in R \text{ and } t \text{ satisfies } qd_k\} \quad k=1,2$
$TQi_{2/1} = \{t \mid t \in R \text{ and } TARG(t) \text{ satisfies } qi_2\}$
$TQd_{2/1} = \{t \mid t \in R \text{ and } TARG(t) \text{ satisfies } qd_2\}$

Read/write sets for Te's in the transformed schema:

$TEi_1 = TQi_1 - TQd_{2/1}$ $\quad TEd_1 = TQd_1$
$TEi_2 = TQi_2 - TQd_1$ $\quad TEd_2 = TQd_2 - TQd_1$
$TEi_{2/1} = TQi_1 \cap TQi_{2/1}$

Target lists for the Tei's in the transformed schema:

The target lists for TEi_1 and TEi_2 are the same as in original schema. The target list for $Tei_{2/1}$ is the composition of $TARG_{i1}$ with $TARG_{i2}$.

Figure 4-9: OE's, OQ's, TE's and TQ's

4.2 Transformation of an operation pair in series

Given a schema OS,(original schema,) for an operation pair in series, shown in figure 4-7, we can find an equivalent schema TS,(transformed schema), as shown in figure 4-8. In this transformed schema Ted_1, Ted_2, Tei_1, Tei_2, and Tei_3 are the transformed operations whose readsets and writesets and target list functions are defined in figure 4-9. This schema evaluates the qualifications based on the original relation R, to get TQd_1, TQi_1, $TQd_{2/1}$, $TQi_{2/1}$, TQi_1, and TQi_2. Using these sets of TIDs, the readsets and writesets of the operation can be determined. The validity of the transformation is proved formally below. Here, we attempt an intuitive justification. (The reader will find it helpful in following this explanation if he/she refers to the example in figure 4-10.) TEd_1 is the same as TQd_1, since both TQd_1 and OQd_1 select tuples from the original state of the relation R. TEi_1 is the set of tuples in TQi_1 and not in $TQd_{2/1}$, because the tuples in $TQd_{2/1}$ are subsequently deleted in step ed_2 of OS. Hence, those tuples which are inserted and subsequently deleted in OS are not inserted at all in TS. Consequently, TEd_2, the set of tuples to be deleted from R by ed_2, are only those which were not already deleted in step ed_1. TEi_2 is the set of tuples in R from which the step ei_2 creates new tuples. So, TEi_2 consists of only those tuples which were not deleted in step ed_1. $TEi_{2/1}$ is the set of tuples inserted by step ei_2 based on tuples inserted by ei_1; the computation performed in this step is the composition of computation specified by the target lists of the two operations. Thus, the transformed schema performs the same deletons and insertions on R as the original schema. And, significantly, the transformed schema allows greater parallelism amongst the qualification steps, which, by our assumption, are more time consuming than the effect steps.

TID	AGE	SALARY	
1	17	10K	Tuples in the old relation
2	16	10K	
3	18	30K	
4	19	30K	
5	17	20K	
6	17	30K	
7	16	30K	
8	16	15K	
9	16	25K	Added Tuples
10	17	25K	
11	18	25K	
12	17	27k	
13	17	27K	
14	17	27K	

qd_1 =(salary = 10K)
qi_1 =(age>= 16 and salary = 30K)
qd_2 =(age = 16)
qi_2 =(age = 17)
$TARG_{i1}$ = (salary = 25K)
$TARG_{i2}$ = (salary = 27K)

R = {1, 2, 3, 4, 5, 6, 7, 8}
R_{os} {3, 4, 5, 6, 10, 11, 12, 13, 14}

OQd_1 = {1,2} OQd_2 = {7, 8, 9} OQi_1 = {3, 6, 7} OQi_2 = {5, 6, 10}

OEd_1 = {1, 2} OEd_2 = {7, 8, 9} OEi_1 = {9,10,11} OEi_2 = {12,13,14}

TQd_1 = {1, 2} TQd_2 = {2,7,8,9} TQi_1 = {3,6,7} TQi_2 = {1, 5, 6}

$TQi_{2/1}$ = {7} $TQd_{2/1}$ = {9}

TEd_1 = {1,2} TEd_2 = {7, 8} TEi_1 = {3, 6} TEi_2 = {5, 6}

$TEi_{2/1}$ = {2}

Fig 4.10: An example of the transformation of an operation pair

To show the validity of the transformation, we first show that the transformed schema is determinate; i.e. all computations for this schema are equivalent. For this we make this following observation, which are proved in the Appendix.

Ted's and Tei's are said to be **confict free** if the writesets of Ted's are mutually disjoint and pairwise disjoint with the read/write sets of Tei's; the readsets of Tei's are pairwise disjoint with the writesets of Tei's; and the writesets of Tei's are mutually disjoint.

Lemma 4-1: $TEd_1, TEd_2, TEi_1, TEi_2$ and TEi_3 are conflict free.

Proposition 4-1: If TEd_1, TEd_2, TEi_1, TEi_2, and TEi_3 are conflict free then the transformed schema is determinate.

This proposition has been proved by Keller [Kell73] for his general parallel program schemata. Intuitively, this follows from the observation that the disjointness property ensures that no tuple is updated by two steps executing in parallel.Lemma 4-1 and Keller's proposition lead to the following theorem.

Theorem 4-1: The transformed schema is determinate.

Having shown that TS is determinate, we now have to show the equivalence of the two schemata to prove the correctness of the transformation. For this we must prove that every tuple $t \in R \cup R_{OS}$ (see figure 4-1) that was deleted, inserted, or remained unchanged in the original schema, was correspondingly deleted, inserted, or remained unchanged in the transformed schema. This is stated in the following sequence of lemmas (the proofs of which appear in the Appendix).

Lemma 4-2: If a tuple $t \in R$ is deleted by OS, then t is also deleted by the transformed schema TS.

Lemma 4-3: If a tuple $t \in R_{OS}$ is inserted by the original schema OS, then t is also inserted by the transformed schema TS, and $t \in R_{TS}$.

Lemma 4-4: If a tuple $t \in R$ is not updated by OS, then t is not updated by TS.

Let $U_{OS} = R||TARG_{i1}(OEi_{i1})||TARG_{i2}(OE_{i2})$,
and $U_{TS} = R||TARG_{i1}(TEi_{i1})||TARG_{i2}(TEi_{i2})||TARG_{i3}(TEi_{i3})$.

Thus, U_{OS} and U_{TS} are the universes for the tuples in relations R_{OS} and R_{TS} respectively.

Lemma 4-5: If a tuple $t \in U_{OS}$ and $t \notin R_{OS}$ then $t \in U_{TS}-R_{TS}$.

Lemma 4-6: $U_{TS} \subseteq U_{OS}$

Lemma 4-7: $(U_{TS}-R_{TS}) \subseteq (U_{OS}-R_{OS})$.

Theorem 4-2: $R_{OS} = R_{TS}$

Proof: To prove this we discuss the following cases:

case 1: $t \in R-R_{OS} \rightarrow t \in R-R_{TS}$ (from lemma 4-2)

case 2: $t \in R_{OS}-R \rightarrow t \in R_{TS}-R$ (from lemma 4-3)

case 3: $t \in R \cap R_{OS} \rightarrow t \in R \cap R_{TS}$ (from lemma 4-4)

case 4: $t \in U_{OS}-R_{OS} \rightarrow t \in U_{TS}-R_{TS}$ (from lemma 4-5)

case 5: $t \in U_{TS}-R_{TS} \rightarrow t \in U_{OS}-R_{OS}$ (from lemma 4-7)

From the above five cases it follows that $R_{OS} = R_{TS}$. [QED]

Thus we have shown that the transformed schema, TS, and the original schema, OS, are equivalent.

4.3 Generalized transformation

We have shown how to transform a pair of operations into a schema having greater parallelism. The transformed schema is not quite in the canonical form of an operation (figure 4-6) because step ei has more than one target list. Thus, if this operation pair is part of a bigger schema and we want it to participate in further transformations with successor nodes, the above transformation cannot be directly used. We now show how to generalize the transformation. First we generalize the concept of an operation pair. This generalization is shown in figure 4-11. The first stage has k insert nodes

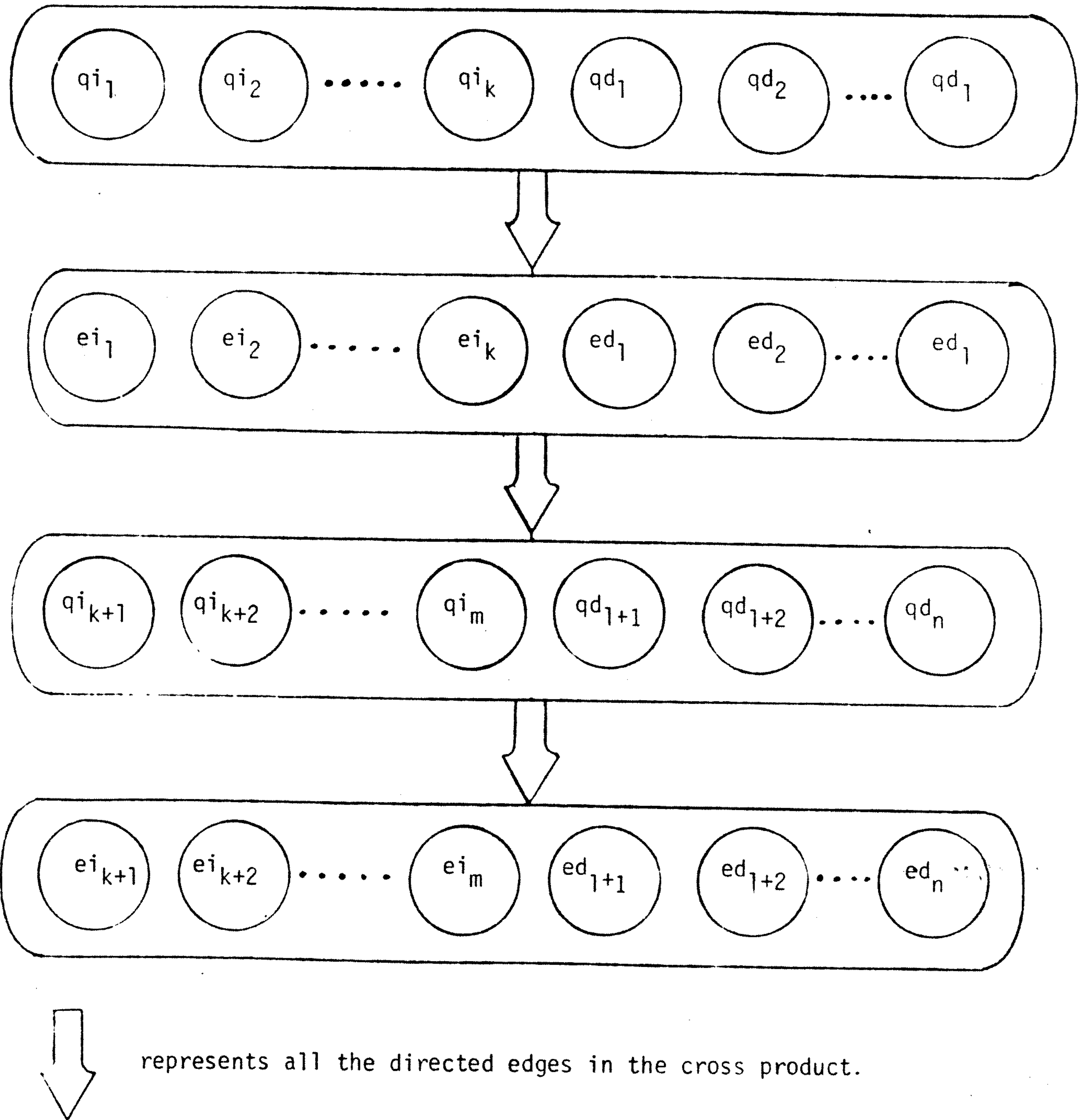
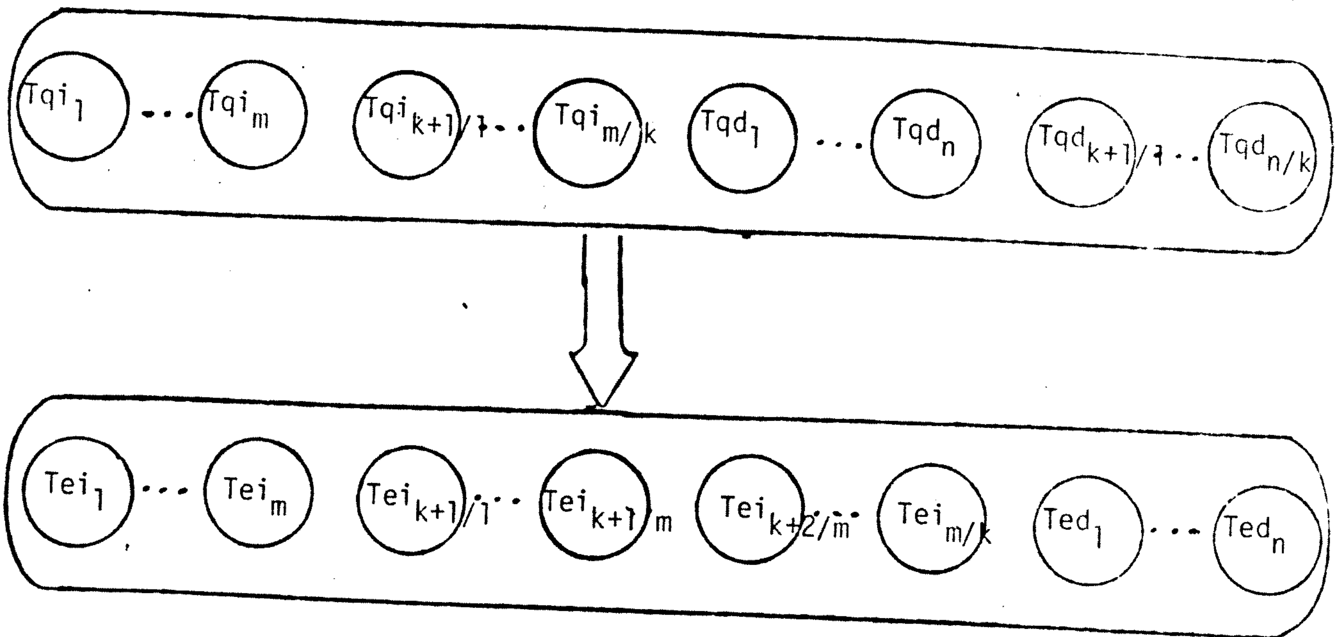


Fig. 4-11: Schema for generalized operation pair



$TEi_a = TQi_a - \bigcup_{b=l+1}^{n} TQd_{b/a}$ $\quad a = 1,2,\ldots k$

$TEd_a = TQd_a$ $\quad a = 1,2,,,,l$

$TEi_a = TQi_a - \bigcup_{b=1}^{l} TQd_b$ $\quad a = k+1,k+2,\ldots m$

$TEd_a = TQd_a - \bigcup_{b=1}^{l} TQd_b$ $\quad a = l+1,l+2,\ldots,n$

$TEi_{a/b} = TQi_b \cap TQi_{a/b}$ $\quad a = k+1,k+2,\ldots,m$; $b = 1,2,\ldots,k$

Fig. 4-12: Transformed schema for generalized operation pair along with the read/write sets.

and l delete nodes. The second stage has m insert nodes and n delete nodes. The transformation and the associated read/write sets are given in figure 4-12. The proof of the previous section can be extended in a straightforward manner for this generalized transformation and does not provide any new insight into the problem. (Further, the proof is messy because of the proliferation of subscripts and is omitted here.) The intuitive justification for the read/write sets is the same as before. TEd_a, a=1,2...l, are the same as TQd_a since both TQd_a and OQd_a select tuples from the original state of R. TEi_a, a=1,2,...k, is the set of tuples in TQi_a and not in $TQd_{b/a}$ for any b $\in$ {l+1,l+2,...n}, because the tuples in $TQd_{b/a}$ are to be subsequently deleted in the second stage of OS. Hence, those tuples which are inserted and subsequently deleted in OS are not inserted at all in TS. Consequently, TEd_a, a=l+1,l+2,....n, the sets of tuples to be deleted from R contain only those tuples that were not already deleted by the first stage. TEi_a, a=k+1,k+2,...m, are the tuples in R from which the steps ei_{k+1}, ei_{k+2}, ...ei_m created new tuples. So TEi_a consists of only those tuples which were not deleted in the first stage. As before, $TEi_{a/b}$, a=k+1,...,m; b=l+1,...,n, is the set of tuples that were inserted in the second stage based on tuples that were inserted in the first stage. Thus the transformed schema performs the same insertions and deletions as the original schema, but has greater parallelism. Furthermore, the transformed schema is in the generalized canonical form of figure 4-11, and so can be used in subsequent transformations.

4.4 Algorithm for transformation

Suppose that the data base machine has k processors. Once the scheduler has constructed a schema representing minimal precedence constraints, it now remains to assign available processors to execute the nodes of the DAG. To maximize processor utilization, it is important to ensure that at every point in time, as many of the k processors as possible are busy. We shall show how to transform the schema to meet this objective.

We have to select k nodes to execute on the k processors. Assuming that each node takes unit time to execute, all k nodes will complete at the same time. (This assumption is relaxed in the section 4-5). At some point in this process, if there are $m<k$ nodes ready for execution, then we can use the generalized transformation developed above to obtain k nodes in parallel. To do this, let n be the number of nodes that can be enabled for execution after the m nodes have been executed. From these n nodes choose $n'=\text{minimum}(n,k-m)$ nodes. Then we can view the DAG as shown in figure 4-13. We see that there are no edges from level 1 to level 0, and from level 2 to either level 1 or 0. To this graph, add edges between every node at level 0 to every node at level 1 (this is not necessarily how the algorithm will be implemented). It is obvious that the graph is still acyclic and the added edges do not contradict any existing precedence constraints. Now the set of nodes at level 0 and level 1 conforms to the generalized canonical form of figure 4-11. So we can apply the generalized transformation to get $m+n'$ nodes in parallel; the transformed graph is shown in figure 4-14. If $m+n'<k$, then this process of transformation can be repeated until k parallel nodes are available.

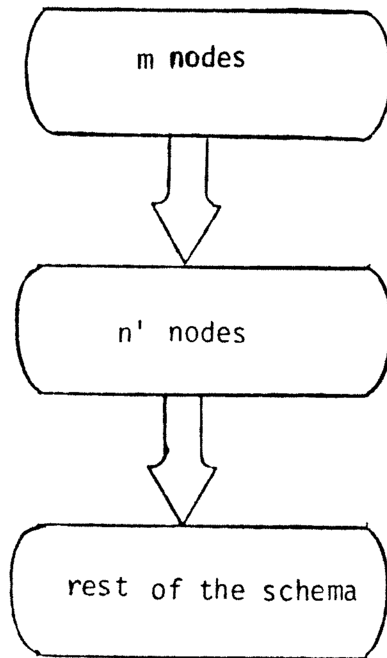


Fig. 4-13: Original DAG

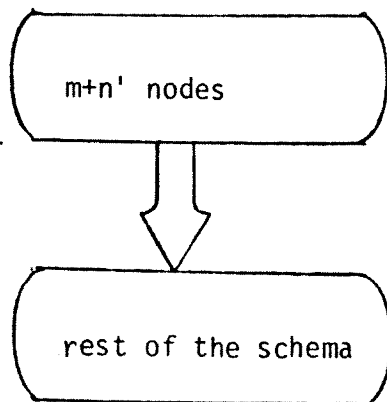


Fig. 4-14: Transformed DAG

4.5 Implementation considerations

Two main assumptions were made in developing the above theory. These were:

1. Only constants were allowed as terms in a target list;

2. all inserted tuples had new TID's.

Both these assumptions were used to ensure that the ed's and ei's were conflict free, i.e. the write sets of ed's were mutually disjoint and pairwise disjoint with the read/write sets of ei's; the readsets of the ei's were pairwise disjoint with the writesets of ei's; and the writesets of ei's were mutually disjoint. We show here that we can relax these assumptions by adopting the following implementation. Associated with each tuple are two flags: delete flag and an insert flag, whose use is described below. Step ed sets the delete flags of all those tuples which are selected for deletion but do not have their insert flags set. These tuples are then deleted during the execution of the next transaction system. Step ei always creates new tuples for insertation and sets their insert flags. If ei is part of a modify operation, then it reuses the old TID; otherwise, it generates a new TID. (This implies that at any point in time, there might be two versions of a tuple both having the same TID.) In reading an existing tuple x, in order to compute a tuple y for insertation, ei uses that version of x which does not have its insert flag set. Further, the insert flag is reset in these tuples at the beginning of the next transaction system.

So any step ed writes only delete flags, and therefore the writesets of ed's are pairwise disjoint with the read/write sets of ei's; we already know from the transformation that the writesets of ed's are mutually disjoint. The insert flag ensures that the readsets of ei's are pairwise disjoint with the writesets of ei's. Creating new tuples ensures that the writesets of ei's are

mutually disjoint. Therefore, the theory of sections 4-2 and 4-3 still holds.

Thus, these transformations can be used repetetively to reduce the diameter of the schema. But it is clear that reduction in the diameter may not be without cost: the number of nodes increases and so does the complexity of each node. It might be more appropriate to pick an optimality measure that takes into account the processing costs of the nodes. Several cost measures for query processing have been proposed in the literature [HY79,Yao79]. These are based on physical parameters such as file sizes, attribute selectivities, storage and access methods. Given any cost measure that imposes a total ordering on the set of schemata, we apply the schema transformations described in this section only if it is beneficial to do so, i.e. only if the estimated cost of the transformation is less than that of the original schema.

5. CONCLUSION

In this paper, we developed a parallel program schema model of transaction systems for parallel database machine.The concept of serializability, which is generally accepted as the correctness in the existing concurrency control theory for the sequential model, was extended to our model. We proposed a two-step technique for producing correct and highly parallel schedules: first, obtain a schema that imposes a minimal set of precedence constraints on correct executions; then, transform the schema using semantic information to increase parallelism. Although the model deveoped in this paper is theoretical, we believe it to be of practical utility -- the proposed scheduling can be applied to any MIMD machine such as DIRECT [DeW78].

Several interesting performance related questions may be posed here. We described the scheduler as a single, centralized process. Will this become a bottleneck? Alternatively, given ample resources and the parallelism inherent

in the system, will it be beneficial to partition the system and distribute the scheduling activity over several processes. Our theory is independent of whether the scheduler is centralised or distributed. Further, we have implied a batched mode of operation for the machine. Each transaction system can be thought of as a batch. This has the advantage that while one transaction system is being executed, the scheduler can be working in parallel on the next transaction system. Clearly, the selection of transactions to comprise a transaction system is a crucial factor affecting performance. An alternative to batching is to dynamically schedule transactions as they arrive. Will this improve performance? Simulation studies or queueing analysis can provide the answers to these questions.

The transformation presented in section 4 produces nodes that must be capable of evaluating arbitrarily complicated set expressions. The complexity of some of these nodes may be reduced by refining the nodes (i.e. replacing each by a more detailed subgraph) and then detecting common subexpressions accross nodes of the subgraphs. As we pointed out before, a cost based on physical database parameters, must be attached to each node. When this is done, it can be determined when it is beneficial to transform a given schema.

Lastly, in section 4 we ignored the problem of eliminating duplicate tuples when an insert or modify operation is executed. We treat this as a special case of integrity checking. Integrity checking could be implemented as part of the effect step of an update operation. However, a more intriguing possibility is to use query modification (as suggested in [ston75]), together with the schema transformation of section 4, to perform integrity checking in parallel with the execution of the update. (for example, tuples which are being duplicated can be flagged for subsequent deletions.) Working out the details of this modifcation is a topic of future research.

REFERENCES

BSR80 P.A.Bernstein, D.W.Shipman,J.B.Rothnie, "Concurrency control in a System of Distributed Databases (SDD-1)" ACM TODS, vol 5, no. 1, 1980.

BSW79 P.A.Bernstein,D.W.Shipman,W.S.Wong, "Formal Aspects of Serilaizability in Database Concurrency control", IEEE-TSE, vol. 5, no. 3, 1979, pp177-187

DeW78 D.J.DeWitt, "DIRECT - A Multiprocessor Organization for Supporting Relational Database Management Systems" Proc. of the 5th Annual Symposium on Computer Arch., Apr. 78, pp182-189

EGLT76 K.P.Eswaren, J.N.Gray, R.A.Lorie, I.I.Traiger, "On the Notions of Consistency and Predicate Locks in a Relational Database System", CACM, vol 19, no. 11, 1976

GOU80 M.Gouda, "Simultanity in Distributes Databses", Technical Report, Dept. of Computer Sciences, Univ. of Texas, Austin, TX, Oct 1980.

HSW75 G.D.Held, M.R.Stonebraker,E.Wong, "INGRES - A Relational Database System", Proc. AFIPS NCC, 1975, pp409-416

HY79 A.R.Hevner, S.B.Yao, "Query Processing in Distributed Database System", IEEE-TSE, vol. 5, no. 3, 1979, pp177-187

KD81 R.Krishnamurthy, U.Dayal, "Complexity of Scheduling Transactions in a Highly Parallel Data Base Machine", Technical Report, TR-169, Dept. of Computer Sciences, Uniov. of Texas, TX, Mar. 81.

Kell73 R.M.Keller, "Parallel Program Schemata and Maximal Parallelism. Part 1: Fundamental results", JACM, 1973, vol. 20, no. 4, pp696-710

KP79 H.T.Kung, C.H.Papadimitriou, "An Optimality Theory of Concurrency Control for Databases", Proc. of 1979 SIGMOD conf., Boston, Mass., May 1979.

Papa79 C.H.Papadimitriou, "Serializability of Database Updates", JACM, vol. 26, no. 4, 1979, pp631-653

PBR77 C.H.Papadimitriou, P.A.Bernstein, and J.B.Rothnie, "Some Computational Problems related to database Concurrency Control", Proc. of Theoretical Computer Science, Waterloo, Aug77.

Ston75 M.R.Stonebraker, "Implementation of Integrity Constraints and Views by Query Modification", Proc. of ACM-SIGMOD Intl. Conf. on Management of Data, San Jose, 1975, pp65-78

Yao79 S.B.Yao, "Optimization of Query Evaluation Algorithms", ACM-TODS, vol. 4, no. 2, 1979

I. Appendix: PROOFS

Lemma 3-1:given $x \in COMP(G)$, $y \in COMP(G_x)$, if x has no dead opeations then y has no dead operaions.

Proof: Let $x = x_1x_2....x_n$ and $y=y_1y_2...y_n$. Let $y_j \in y$ be such that y_{j+1}, y_{j+2}, y_n are live, and y_j is dead. Find j′ such that $x_{j'}=y_j$. Obviously there does not exist m such that $(y_j,y_m) \in RF_y$, but there does exists k′ such that $(x_{j'},x_{k'}) \in RF_x$. Find k such that $y_k=x_{k'}$. But, k<j is not possible because $(x_{j'},x_{k'}) \in RF_x$ and therefore $(x_{j'},x_{k'})$ is a precedence constraint in G_x. So $(y_j,y_k) \in E_x$. From this it directly follows that j<k. • From the definition of liveness we can conclude that

$$\exists_l [(j<l<k) \wedge (\{W(y_j) \cap R(y_k)-W(y_l)\}=\emptyset)] \text{ or } W(y_j) \cap W(y_l) \neq \emptyset$$

Find l′ such that $x_{l'}=y_l$. But, j′<l′<k′ is not possible because $x_{k'}$ reads from $x_{j'}$. And if j′>l′ or k′<l′ then there will be an interferes edge in G_x to impose that same order in y. Thus,there can be no dead operation in y. [QED]

Lemma 3-2: Given $x \in COMP(G)$, $y \in COMP(G_x)$ it follows that $RF_x = RF_y$

Proof: Let $x=x_1x_2...x_n$, and $y=y_1y_2...y_n$. Find a Y_j such that y_j reads the value in cell m written by y_k, and the coresponding $x_{j'}$ $(=y_j)$ does not read the value in cell m written by $x_{k'}$ $(=y_k)$. Let $x_{j'}$ read the value in cell m written by $x_{l'}$. Find l such that $x_{l'}=y_l$.

As $x_{j'}$ reads from $x_{l'}$, so $\quad l'<j'$ ....(1)
As y_j reads from y_k , so $\quad k<j$ ....(2)

If k′>j′, then, by definition, $(x_{j'},x_{k'})$ is an interferes edge, and consequently, a precedence constraint in G_x. Then, k<j as in (2) is not possible. So, $\quad k' \not> j'$ ....(3)
If l′<k′<j′, then $x_{j'}$ will not read from xl′ as assumed. so $\quad k'<l'<j'$ ....(4)
using (1) and precedence constraints in G_x, $\quad l \not> j$ ....(5)
If k<l<j, then y_j will not read from y_k as assumed. so $\quad l<k<j$ ....(6)

From (4) we note that there will be an interferes edge $(x_{k'},x_{l'})$ in G_x and

this order is not retained in y, as given in (6), this contradicts the precedence constraint in G_x.

Hence, we can conclude that if y_j reads m from y_k, then $x_{j'}$ reads m from $x_{k'}$, for all m in the domain of y_j. In other words $RF_x^m \subseteq RF_x^m$ for all $m \in M$

Now we show that if x_j reads m from x_k, then $y_{j'}$ $(=x_j)$ reads m from $y_{k'}$ $(=x_k)$ for all x_j and every $m \in R(x_j)$. If this is not true, then there exists an x_j that reads m from x_k, and $y_{j'}$ does not read m from $y_{k'}$. Then $y_{j'}$ reads m from some other operation, which contradicts the above claim that $RF_x^m \subseteq RF_x^m$. So it must be that $RF_x^m \subseteq RF_x^m$. Hence we have shown that, $RF_x^m = RF_x^m$ for all $m \in M$.

Lemma 2.1: Given a SR-schema G_p of a serial execution SX_p, every execution resulting from G_p is serializable.

Proof: From lemma 3-3 it follows directly that every computation sequence in $COMP(G_p)$ is serializable.

We know that every edge (i.e. precedence constraint) in G_p is also an edge in any execution X resulting from G_p. Therefore every topological sort sequence of X must also be a topological sort sequence of G_p. So,

$$COMPUT(X) \subseteq COMP(G_p)$$

Therefore we can conslude that every topological sort sequence of COMPUT(X) is serializable to SX_p.

Lemma 4-1: $Ted_1, Ted_2, Tei_1, Tei_2$ and Tei_3 are mutually disjoint.

Proof: First, observe that every inserted tuple is a new tuple. From this it directly follows that $(TEd_1 \cup TEd_2) \cap (TEi_1 \cup TEi_2 \cup TEi_3) = \emptyset$ and TEi_1, TEi_2, and TEi_3 are mutually disjoint. From the definition of TEd_1 and TEd_2 it follows that $TEd_1 \cap TEd_2 = \emptyset$. QED

Lemma 4-2: If a tuple $t \in R$ is deleted by OS, then t is also deleted by the transformed schema TS.

Proof: <u>case 1 ($t \in OQd_1$)</u>: As $TEd_1 = OQd_1$, t is also deleted by TS.

<u>case 2 ($t \in OQd_2$)</u>: This implies that t was not deleted by step ed_1; i.e. $t \notin OEd_1$. From the definition of TQd_2, we know that $t \in TQd_2$, and as $TEd_1 = TQd_1 = OQd_1 = OEd_1$; so $t \in TEd_2$ and is deleted by step Ted_2.

From the above two cases we know that t will be deleted and from the determinacy of TS we are assured of no other updates on t. QED

Lemma 4-3: If a tuple $t \in R_{OS}$ is inserted by the original schema OS, then t is also inserted by the transformed schema TS, and $t \in R_{TS}$.

Proof: Let t_1, t_2 be tuples such that $t = TARG_{i1}(t_1)$ and $t = TARG_{i2}(t_2)$

depending on whether ei_1 or ei_2 inserted the tuple t in OS.

case 1 ($t_1 \in OQi_1$ and $t_1 \notin OQd_2$): $t_1 \in OQi_1$ implies $t_1 \in TQi_1$ (by definition). If $t_1 \in TQd_{2/1}$ then the tuple t inserted in step ei_1 will be selected by qd_2; i.e. $t \in OQd_2 = OEd_2$. Then t cannot be a tuple in R_{OS} as step ed_2 would have deleted it. So $t_1 \notin TQd_{2/i1}$. It follows that $t_1 \in TEi_1$. So from lemma 4-1 we can conclude that in both the schemata $t = TARG_{i1}(t_1)$ is inserted.

case 2 ($t_2 \in OQi_2$): If $t_2 \in TQi_2$ then $t_2 \in TEi_2$ because if $t_2 \in TQd_1 = Qd_1$ then t_2 could not have been selected by qi_2, since it would have been deleted by ed_2; but this would contradict $t_2 \in OQi_2$. If $t_2 \notin TQi_2$ then we observe from the precedence constraints of OS that $t_2 \in TARG_{i1}(OEi_1)$. This implies that there exists a tuple $t_1 \in R$, such that $t_2 = TARG_{i1}(t_1)$ and $t_1 \in OQi_1$. From the definition of $TQi_{2/i1}$, if $t_1 \in R$ and $t_2 = TARG_{i1}(t_1) \in OQi_2$ then $t_1 \in TQi_{2/i1}$. As we already know, $t_1 \in OQi_1$ implies $t_1 \in TQi_1$, so $t_1 \in TQi_1 \cap TQi_{2/i1}$. Thus, OS and TS insert the same tuple $TARG_{i2}(t_2) = TARG_{i2}(TARG_{i1}(t_1))$. Once again, from Lemma 4-1 we can conclude that no other update was done on that tuple.

QED Lemma 4-4:If a tuple $t \in R$ is not updated by OS, then t is not updated by TS.

Proof: If $t \in R$ and $t \in R_{OS}$, then t is not a tuple that was inserted by OS. Further, since, t is not updated by OS, $t \notin OQd_1$ and $t \notin OQd_2$. $t \notin OQd_1$ implies $t \notin TQd_1 = TEd_1$. If $t \in TQd_2$, then $t \in OQd_2$ because $t \in R$ and $t \notin TQd_1$. So $t \notin TQd_2$. Therefore, $t \notin TEd_2$. That t is not updated in TS follows from the mutual disjointness property.

QED

Lemma 4-5:If a tuple $t \in U_{OS}$ and $t \notin R_{OS}$ then $t \notin R_{TS}$.

Proof: If $t \in R$ then by Lemma 4-2 $t \notin R_{TS}$. If $t \in TARG_{i2}(OEi_2)$ then $t \in R_{OS}$; so this lemma is vacuously true. If $t \in TARG_{i1}(OEi_1)$ and $t \notin R_{OS}$ then it

follows that $t \in OQd_2=EQd_2$; so there exists a t' such that $t = TARG_{i1}(t')$ and $t' \in TQd_{2/i1}$ and from the definition of TEi_1 we see that $TARG_{i1}(t')$ was never inserted. so $t=TARG_{i1}(t') \notin R_{TS}$. QED

Lemma 4-6:$U_{TS} \subseteq U_{OS}$

Proof: $TEi_1 \subseteq OEi_1$ directly follows from definition. If there is a tuple $t \in TEi_2$ such that $t \notin OEi_2$ then it follows that $t \in R$ and $t \notin R-OEd_1 \cup OEi_1$; hence, $t \in OEd_1$. From the definition of TEi_2 we can conclude that $t \notin OEd_1$, which is a contradiction. Therefore, $TEi_2 \subseteq OEi_2$. Finally, for some tuple $t \in R$, let $TARG_{i2}(TARG_{i1}(t)) \in TARG_{i3}(TEi_3)$.From the definition we see that $t \in TEi_3$; that is $t \in TQi_1 \cap TQi_2$. $t \in TQi_1$ implies that there exists a tuple t' such that $TARG_{i1}(t)=t'$ and t' satisfies qi_2; i.e. $t' \in TQ-(i2)$. As $t \in OQi_1$, we can infer that $TARG_{i1}(t) \in (R-OEd_1) \cup TARG_{i1}(OEi_1)$. Therefore, $TARG_{i1}(t) \in OQi_2$ since $TARG_{i1}(t)=t'$ satisfies qi_2. This implies that ei_2 inserts a tuple $TARG_{i2}(t') = TARG_{i2}(TARG_{i1}(t))$; that is, $TARG_{i2}(TARG_{i1}(t)) \in R_{OS}$. Hence, $TARG_{i3}(TEi_3) \subseteq TARG_{i1}(OEi_1) \cup TARG_{i2}(OEi_2)$.

The above inferences imply that $TARG_{i1}(TEi_1) \cup TARG_{i2}(TEi_2) \cup TARG_{i3}(TEi_3) \subseteq TARG_{i1}(OEi_1) \cup TARG_{i2}(OEi_2)$; i.e. $U_{TS} \subseteq U_{OS}$. QED

Lemma 4-7: $(U_{TS}-R_{TS}) \subseteq (U_{OS}-R_{OS})$.

Proof: Assume that there exist a tuple $t' \in U_{TS}-R_{TS}$ such that $t' \notin U_{OS} \vee t' \in R_{OS}$. From lemma 4-6 we see that $t' \in U_{TS}$ implies that $t' \in U_{OS}$. So it follows that $t' \in R_{OS}$. From lemma 4-2 and lemma 4-3 we see that $t' \in R_{OS}$ implies that $t' \in R_{TS}$ which contradicts the assumption. Hence, it follows that $t' \in U_{OS}-R_{OS}$. QED