マルチメディア通信と分散処理 37－10
（１９８８．５．２０）

# Prologトランザクションの並行実行

滝沢 誠　　宮島 勝己

東京電機大学理工学部

本論文では、分散したデータベースシステムに対するトランザクションをPrologにより記述し、これを複数のデータベースシステムにより並行実行する問題を考える。まず、Prologによりトランザクションを記述するために新たにWrite, Begin, Commit, Abortという述語記号を導入する。次に、トランザクション内の基本式のなかで意味のある順序として、競合と流関係を導入する。これらの順次関係を保存するようにトランザクションを、各データベースシステム毎に実行できる部分トランザクションに分割する。このとき、導出の前進時に通信についてのデッドロックが生じない方法を示すと同時に、後戻り時に生じるデッドロックを検出し除去する方法を与える。本論文により、異種のデータベースシステムからなる分散型データベースシステムに対して、共通のPrologを用いてトランザクションを記述し、これを並行実行できる。

# PARALLEL EXECUTION OF PROLOG TRANSACTIONS

Makoto TAKIZAWA and Katsumi MIYAJIMA

Department of Information and Systems Engineering
Tokyo Denki University
Ishizaka, Hatoyama-cho, Saitama 350-03, Japan
Tel. 0492-96-2911 ext.246
Fax. 0492-96-0501

Information systems are composed of multiple heterogeneous database systems interconnected by communication networks. In our system, database systems are viewed to be fact base systems (FBSs), which provide sets of Prolog ground unit clauses, and retrieval and update operations on them. Transactions composed of multiple update operations on facts are written in Prolog. In the Prolog system, atoms in a goal are selected from left to right. This means read/write operations on facts are sequentially executed. In this paper, we present how to execute the transaction in parallel by multiple FBSs. In the sequence of atoms in the Prolog transaction, we define two types of meaningful sequences, i.e. conflict and flow ones. We present how to decompose transactions into subtransactions, each of which is locally executed by one FBS, so as to preserve the meaningful sequences in the transaction. One problem in concurrent execution is that communication deadlock may occur. We show that any parallel execution of the subtransactions is free of deadlock on send-receive relations.

## 1. INTRODUCTION

Information systems are composed of database systems interconnected by communication networks, which users can manipulate independently of the heterogeneity and distribution. Prolog [CLOC,KOWA] have advantages to conventional database languages like SQL[DATE], i.e. data structures and procedures are defined recursively and uniformly. In our system[TAK87a,88a,b], every database system is viewed to be a fact base system (FBS) which provides a set of facts, i.e. ground unit clauses, and operations on them. The heterogeneous database systems are manipulated in Prolog as if they were FBSs. How to provide Prolog views on the relational database systems[CODD] are discussed in [ULLM etc.]. Also, Prolog views on the network database systems[CODA] are discussed in [TAK87a].

In our system, users write transactions which manipulate facts in multiple FBSs in Prolog. Transactions[GRAY] are procedures composed of multiple update operations on facts, and are units of atomic executions. In the Prolog systems, atoms in a goal are selected from left to right, i.e. the atoms are sequentially executed. In the atom sequence of the Prolog transaction, two types of meaningful sequences, i.e. conflict and flow ones are defined in [TAK88b]. This paper presents how to decompose the transaction T into subtransactions, each of which references facts in one FBS, so as to preserve the meaningful sequences in T, and which are concurrently executed by the FBSs. Recently, Prolog programs have been tried to be executed in parallel[CLAR,SHAP,UEDA etc.]. In addition to input-output relation[CLAR,SHAP] among atoms, read-write and write-write relations named conflict ones are introduced in this paper. Update problems of a single database system by Prolog have been discussed by [NAIS]. In this paper, update problems of multiple database systems are discussed. One problem in the concurrent execution of subtransactions by multiple FBSs is how to resolve communication deadlocks. We assume that each FBS is a sequential machine. We define two kinds of deadlocks, i.e. ones occur in forwarding deductions and the others in backtracking. In this paper, we show a decomposition method which never implies forward deadlocks on send-receive synchronization, and control method which detects deadlocks and breaks them in backtracking.

In chapter 2, Prolog transactions are presented. In chapter 3, we define what is a meaningful sequence of atoms in the transaction. In chapter 4, we discuss how to decompose Prolog transactions into subtransactions. In chapter 5, communication deadlock problems are discussed.

## 2. PROLOG TRANSACTIONS

A fact base (FB) is a Prolog view of the conventional database system. The FBs for various database systems are presented in [TAK87a,88a]. Suppose that every database system provides its FB, i.e. it can be viewed as a collection of ground unit clauses. Here, we make the following assumptions.

[Assumptions] 1) Every two FBs have no common predicate symbols.
2) Let $M_k$ be a model of the FB in the $FBS_k$. Every $M_k$ has the same domain D and the same constant mapping. That is, the same constant symbol in different FBs denotes the same individual in D.
3) Every fact has a unique fact identifier (fid).□

In order to write transactions in Prolog, we introduce new predicate symbols, Begin, Commit, Abort, and Write. Their procedural meanings are as follows.

[Procedural Meanings] 1) When Begin is selected, the transaction starts.
2) When Commit is selected, the transaction completes. After that, we cannot backtrack to resolutions from the selection of Begin to this Commit. Facts written by the transaction are made permanent, i.e. updated facts can be seen by other transactions and survive various failures of the FBS.
3) When Abort is selected, the resolution fails, and the resolutions from it to the last Begin are backtracked, i.e. the update effects by the transaction are erased from the FB.

4) When Write(P,K,$X_1$,...,$X_m$), where P is a predicate symbol in the FB, and K and $X_k$ are terms (k=1,...,m), is selected,

4-1) if K is instantiated by @p and $X_k$ by $a_k$ (k=1,...,m), the fact P(@p,...) is replaced with P(@p,$a_1$,...,$a_m$),

4-2) if every $X_k$ is instantiated by $a_k$ (k=1,...,m) and K is not, P(@p,$a_1$,..., $a_m$) with new fid @p is appended to the FB and K is instantiated by @p,

4-3) if K is instantiated by @p and every $X_k$ is not (k=1,...,m), P(@p,...) is deleted from the FB.

5) When a fact atom P($E_1$,...,$E_m$) is selected, a fact unifiable with it is read from the FB, and the uninstantiated variables are instantiated. □

Variables which take fids as values are said to be primary. Write is different from the retract and assert of Prolog[CLOC] in a point that effects of Write are removed from the FB by backtracking.

Suppose there are three fact base systems (FBSs), i.e. $FBS_1$, $FBS_2$, and $FBS_3$. Each includes account information on individuals. Atoms AC1, AC2, AC3 are in $FBS_1$, $FBS_2$, and $FBS_3$, respectively. ACi (A,B,C) means that a person B has money C and its fid is A (i=1,2,3). An example of Prolog transaction which transfers funds of individuals is written as (1).

?- Begin, AC1(A1,N,C1), AC3(A3,N,C3), Write(AC1,A1,N,C1/2+C3/3), AC2(A2,N, C2), Write(AC2,A2,N,C1/2+C2+C3/3), Write(AC3,A3,N,C3/3), Commit. ...(1)

[Definition] A simple transaction T is an ordered goal clause in a form ?- Begin, $A_1$,...,$A_m$, Commit, where each $A_k$ is either a fact, Write, or evaluatable atom (for k=1,...,m). T is written as (AA,<), where AA={$A_1$,...,$A_m$} and $A_j < A_k$ iff $j < k$.□

The procedural meaning of the simple transaction is a sequential execution of write and read operations denoted by Write and fact atoms. It neither includes nondeterminacy nor nesting[MOSS].

### 3. CONFLICT AND FLOW RELATIONS

[Definition] For a variable X and two different atoms A and B, A ◇ B on X (A and B share X) iff they include X.□

Here, if A and B are in different FBSs and A ◇ B on X, ◇ is said to be an intersite sharing relation.

[Definition] For two different atoms A and B in T = (AA,<), A → B (A and B conflict) iff (1) A < B, (2) A ◇ B on a primary variable K, and (3) one of the followings holds:

1) A = p(K,$T_1$,...,$T_m$) and B = Write(p,K, $U_1$,...,$U_n$),

2) A = Write(p,K,$T_1$,...,$T_m$) and B = q(K, $U_1$,...,$U_n$), or

3) A = Write(p,K,$T_1$,...,$T_m$) and B = Write (p,K,$U_1$,...,$U_n$).□

That is, A and B conflict iff at least one of them is Write and they share some primary variable. For example, AC1(A1, N,C1)→ Write(AC1,A1,...) in (1). When A→ B in T, selection sequence of A and B cannot be exchanged. If exchanged, the result is different from T. The conflict relation represents read-write and write-write relation[GRAY].

Next, we define a flow relation which represents an output-input relation among the atoms. An atom A in T is said to be a first atom of a variable X iff A includes X and there is no atom B such that B includes X and B < A.

[Definition] For every atoms A and B in T =(AA,<), A ⇒ B(A flows into B) iff 1) A < B, 2) A ◇ B on a variable X, 3) A≠ Write, and 4) A is the first atom of X. □

A ⇒ B denotes an information flow from an A's fact into a B's fact. For example, AC1(A1,N,C1) ⇒ Write(AC2,A2,N,... C1/2...) on C1 holds in (1). This means that a value of C1 read from the AC1 is written to the AC2. Like the conflict relation, if A ⇒ B in T, the selection sequence of A and B cannot be exchanged. For two atoms in different FBSs, if A ⇒ B, then it is said to be an intersite flow, otherwise an intrasite one. For example, an intersite AC1(A1,N,C1) ⇒ Write(AC2,A2,N,C2+C1/2+C3/3) and an intrasite AC1(A1,N,C1) ⇒ Write(AC1,A1, N,C1/2+C3/3) hold in (1).

In (1), AC1(A1,N,C1) ⇒ Write(AC1,A1,N,

C1/2+C3/3) and AC1(A1,N,C1) ◇ AC2(A2,N,C2) hold. Here, even if AC1(A1,N,C1) and AC2(A2,N,C2) are exchanged, it is clear that the results are the same.

[Definition] Let A and B be non-Write atoms, and C be Write in T = (AA,<). An implicit flow relation ==> on AA is one such that B ==> C on a variable X if 1) A ◇ B on X, 2) neither C => A nor C => B, and 3) A => C on X.□

For example, AC2(A2,N,C2) ==> Write(AC1, A1,N,C1/2+C3/3) holds in (1).

[Definition] A meaningful ordering relation ≪ $\subseteq$ $AA^2$ is defined as follows; for every A and B in AA, A ≪ B iff 1) A→B, A => B, or A ==> B, or 2) A ≪ C and C ≪ B.□

[Definition] $T_1 = (AA_1, <_1)$ and $T_2 = (AA_2, <_2)$ are said to be equivalent iff $AA_1 = AA_2$ and $\ll_1 = \ll_2$.□

**[Proposition]** If two simple transactions $T_1$ and $T_2$ are equivalent, they produce the same result for the same input.
[Proof] It is sufficient to prove that 1) A => C and 2) B => C are equivalent when A ◇ B and A ◇ C on X, and C is Write. Since neither C ≪ B nor C ≪ A, an instantiation of X which satisfies A and B is flown to C.■

In the Prolog system, every variable is instantiated only once[SHAP]. This means that a single assignment property[ACKE] holds in the resolution. Hence, it is easy to find the meaningful sequences.

## 4. DECOMPOSITION OF PROLOG TRANSACTION

In our approach to executing the transaction, it is concurrently executed by multiple FBSs. In this approach, first, transactions are decomposed into subtransactions, each of which references facts in one FBS. Then, these subtransactions are concurrently executed by FBSs by communicating with each other.

### 4.1 Transaction Graph

[Definition] For a transaction T = (AA, <), a transaction (T) graph G is one obtained by the following procedure:
1) generate a node A for every atom A,
2) for every A→B in T, generate a conflict edge→from the node A to B,
3) for every A=>B or A==>B, if it is an intrasite one, generate an intrasite flow edge A **> B, else an intersite flow edge A ++> B, and
4) for every A ◇ B in T, generate a sharing edge A = B if A and B are in the same FBS, otherwise A == B.□

A T graph for (1) is shown in Fig.1. Since the T graph G shows the meaningful ordering ≪ of atoms in the transaction T = (AA,<), G is written as (AA,≪).

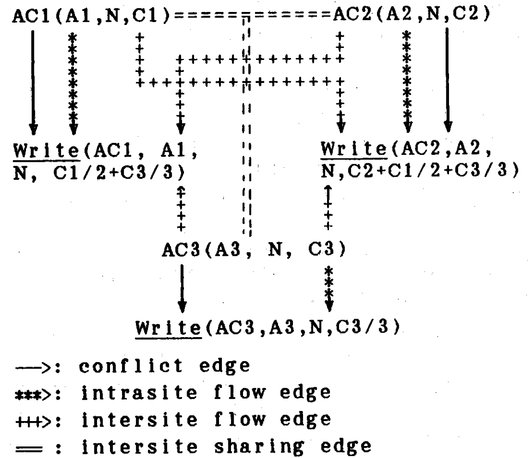


**Fig.1** Transaction Graph

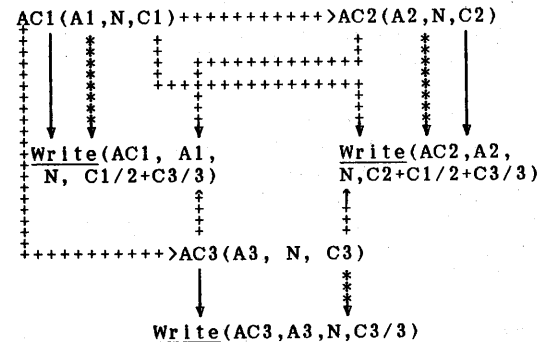


**Fig.2** Transaction Graph

First problem is how to find a best transaction equivalent to T. Since it is difficult to find the best one from the complexity viewpoint, we make the following assumption.

[Assumption] If A◇B on X and A < B in T

$= (AA,<)$, then $A \Rightarrow B$ on X.□

A transaction graph as shown in Fig.2 is obtained.

### 4.2 Decomposition of Transaction Graph

In order to pass the instantiation of variables, communication among subtransactions is required. Here, new predicate symbols Send and Receive are introduced for communication. They have the following procedural meanings.

[Communication Atoms] 1) When Send(X,M) is selected, a variable M is instantiated by X.
2) When Receive(M,X) is selected, the resolution blocks until M is instantiated. When instantiated, X is instantiated by M.□

Here, M is a communication variable. Even if multiple Receives include the same communication variable M, they can be resolved when one Send atom including M is resolved. This is easily realized by the reliable broadcast network[TAK87b,c].

A T graph $G = (AA,\ll)$ of $T = (AA,<)$ is decomposed into subgraphs $S_1,\dots,S_n$ by the following procedure.

[Decomposition Procedure] 1) For each $A \leftrightarrow B$ on a variable X, mark X in A as $X^+$ and X in B as $X^-$ and remove $A \leftrightarrow B$,
2) let each connected subgraph $S_k = (AA_k,\ll_k)$ be a subgraph.□

For example, Fig.3 shows subgraphs $S_1$, $S_2$, and $S_3$ decomposed from G in Fig.2.

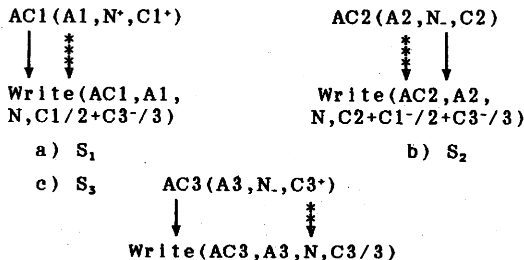


**Fig.3** Decomposed Subgraphs

Now, communication atoms are introduced to the subgraphs. For each subgraph $S_k = (AA_k,\ll_k)$, the subtransaction graph $G_k = (BB_k,\lll_k)$ is constructed by the following procedure.

[Subtransaction Graph]
1) $AA_k \subseteq BB_k$ and $\ll_k \subseteq \lll_k$.
2) For each variable $X^+$ in $S_k$, Send(X,CX) in $BB_k$ and $A \lll_k$ Send(X,CX).
3) For every $X^-$ in $S_k$, Receive(CX,X) in $BB_k$ and Receive(CX,X) $\lll_k A$.
4) If $A \lll_k B$ and $B \lll_k C$, then $A \lll_k C$.□

For example, Fig.4 shows the Hasse diagrams of subtransaction graphs for Fig.3, where each edge → shows $\lll_k$.

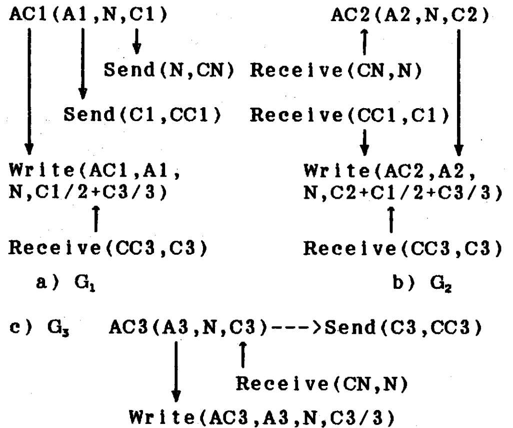


**Fig.4** Subtransaction Graphs.

## 5. COMMUNICATION DEADLOCKS

From a subtransaction graph $G_k = (BB_k,\lll_k)$, let us consider how to obtain a subtransaction $T_k = (BB_k,<_k)$ so as to preserve the meaningful relations $\lll_k$.

### 5.1 Forward Deadlock

Now, let us construct subtransactions $T_k = (BB_k,<_k)$ from $G_k = (AA_k,\lll_k)$, which preserves $\lll_k$, i.e. if $A \lll_k B$, then $A <_k B$. For example, from $G_1$, $G_2$, and $G_3$ in Fig.3, the following subtransactions are obtained. It is clear that each $T_k$ preserves the meaningful relation $\lll_k$ $(k=1,2,3)$.

$T_1$ ?- Begin, AC1(A1,N,C1), Receive(CC3, C3), Send(C1,CC1), Send(N,CN), Write (AC1,A1,N,C1/2+C3/3), Commit. ...(2)
$T_2$ ?- Begin, Receive(CN,N), Receive (CC3,C3), AC2(A2,N,C2), Receive(CC1,

C1), Write(AC2,A2,N,C2+C1/2+C3/3), Send(C2,CC2), Commit. ...(3)

$T_3$ ?- Begin, Receive(CN,N), AC3(A3,N, C3), Write(AC3,A3,N,C3/3), Send(C3, CC3), Commit. ...(4)

The resolution of (2) is executed at $FBS_1$, (3) at $FBS_2$, and (4) at $FBS_3$. When FBSs are forwarding resolutions of them by selecting atoms from left to right, a deadlock occurs, i.e. $T_1$ waits for CC3 from $T_3$, and $T_3$ for N from $T_1$. This type of communication deadlock occurred in the forward resolutions is said to be a forward deadlock. In our approach, the forward deadlock is avoided by ordering communication atoms in subtransactions.

[Ordering Send and Receive] For $G_k = (BB_k, <<<_k)$ of $T = (AA, <)$, if $A <<<_k$ Send, Receive $<<<_k B$, and $A < B$, then Send $<<<_k$ Receive.□

Subtransactions (5)～(7) are constructed from (2)～(4) by ordering Send and Receive.

$T_1$ ?- Begin, AC1(A1,N,C1), Send(C1, CC1), Send(N,CN), Receive(CC3,C3), Write(AC1,A1,N,C1/2+C3/3), Commit. ...(5)

$T_2$ ?- Begin, Receive(CN,N), AC2(A2,N, C2), Receive(CC1,C1), Receive(CC3, C3), Write(AC2,A2,N,C2+C1/2+C3/3), Send(C2,CC2), Commit. ...(6)

$T_3$ ?- Begin, Receive(CN,N), AC3(A3,N, C3), Write([AC3,A3,N,C3/3]), Send(C3, CC3), Commit. ...(7)

**[Proposition]** Any executions of subtransactions decomposed by our procedure are deadlock-free on the forwarding resolution.

[Proof] This is proved in [TAK88b]■

### 5.2 Sequencing Subtransactions

Although the method presented in 5.1 avoids the forward deadlock, it may reduce the chance of getting better sequences of atoms, because $A <<<_k$ Send $<<<_k$ Receive $<<<_k B$ implies $A <<<_k B$. The reverse order of A and B may imply better performance. Among multiple equivalent sequences, one sequence is selected so as to minimize the waiting time by Receive and the access cost to the underlying database systems. In order to achieve the objectives, the following heuristics totally orders atoms for a subtransaction graph $G_k = (BB_k, <_k)$.

[Ordering Unordered Send and Receive]

1) If $A <<<_k B$, then $A <_k B$.
2) If $A <<<_k$ Send, Receive $<<<_k B$, and $A <<<_k B$, then Send $<_k$ Receive.
3) If $A <<<_k$ Send, $A <<<_k B$, and B is not a communication atom, then Send $<_k B$.
4) If $A <<<_k B$, Receive $<<<_k B$, and A is not a communication atom, then $A <_k$ Receive.
5) If $A <<<_k$ Send and $A <<<_k$ Receive, then Send $<_k$ Receive.
6) If $A <<<_k$ $Send_1$, $A <<<_k$ $Send_2$, and $Send_1 < Send_2$, then $Send_1 <_k Send_2$, and
7) If $A <<<_k$ $Receive_1$, $A <<<_k$ $Receive_2$, and $Receive_1 < Receive_2$, then $Receive_1 <_k Receive_2$.
8) If $A <<<_k B$, $A <<<_k C$, B and C are neither ordered nor communication atoms, and cost(B) < cost(C), then $B <_k C$.□

Cost function cost(X) for an atom X gives an expected number of tuples accessed to find a unifiable tuple with X. This is computed based on the statistics[HEVN,TAK88a] on the underlying database. The selection of Receive(M,X) blocks the resolution of the transaction until M is instantiated, i.e. M is received. Hence, it is better to delay the selection of Receive and instead select an atom A such that there is no meaningful ordering among Receive and A. Also, it is better to promote the selection of Send(X,M) as possible, because it may cause blocked subtransactions which wait for M to execute. This is done by (2) and (3). For example, atoms Send(C3,CC3) are moved after AC3 in (7) as shown in (10).

$T_1$ ?- Begin, AC1(A1,N,C1), Send(C1, CC1), Send(N,CN), Receive(CC3,C3), Write(AC1,A1,N,C1/2+C3/3), Commit. ...(8)

$T_2$ ?- Begin, Receive(CN,N), AC2(A2,N, C2), Send(C2,CC2), Receive(CC1,C1), Receive(CC3,C3), Write(AC2,A2,N,C2+ C1/2+C3/3), Commit. ...(9)

$T_3$ ?- Begin, Receive(CN,N), AC3(A3,N, C3), Send(C3,CC3), Write(AC3,A3,N, C3/3), Commit. ...(10)

Last problem is how to select an atom A among ones which includes marked variable X as shown in 4.1. In our heuris-

tics, an atom A whose cost(A) is the minimum is selected as a sender of X and the others as the receivers.

### 5.3 Backward Deadlocks

Suppose that the resolution of Write in (5) fails and the resolution of AC3 fails in (7). Both transactions backtrack to Receives, and $T_1$ waits for CC3 from $T_3$ and $T_3$ waits for CN from $T_1$. That is, deadlock occurs. This type of deadlock is a backward deadlock. Although we avoid the forward deadlock by ordering communication atoms, the backward one is solved by detecting and breaking it according to the following mechanism.

[Backward Deadlock Resolution] Let a current atom in $T_K$ be one which is selected in the deduction at present.
1) On backtracking to Send(X,C), a message FAIL(C) is broadcast to all subtransactions and the resolution fails.
2) On backtracking to Receive(C,X), a message WAIT(C) is broadcast and waits for the arrival of a message at C.
3) On receipt of a message FAIL(C), if an atom Receive(C,X) is included, the messages in C are all cleared. For the current atom A, if Receive(C,X) $<_K$ A, we directly backtrack to the Receive. If communication atoms are included in between A and the Receive, they are resolved according to (1) and (2).
4) On receipt of WAIT(C), a wait-for graph whose nodes are subtransactions and directed edges from A to B indicate that A waits for message from B.□

At 4), if the wait-for graph contains a directed cycle, a backward deadlock occurs. When the deadlock is detected, one edge is selected and it is broken. Let us discuss how to select one edge and break it. Let A and B be atoms in $T_K$. Suppose that the resolution of B fails. The backtrack cost Bcost(B,A) of B to A is defined to be the number of atoms from A to B in $T_K$. Let A be a Send atom such that there is no Send such that A $<_K$ Send $<_K$ B. Here, let bcost(B) be Bcost(B,A). At 2), WAIT(C) with bcost(Receive(C,X)) is broadcast. On receipt of WAITs, the subtransaction $T_J$ whose backtrack cost bcost(B) is the minimum is selected. In $T_J$, we backtrack from B = Receive(C,X) to the Send according to 1)～4). By using the reliable broadcast service [TAK87b,c], each FBS can decide by itself that a backtracking deadlock occurs and which subtransaction is backtracked.

### 5.4 Correctness

Now, we show the correctness of our decomposition of the transaction.

[Definition] Let $T_K$ be a subtransaction $(BB_K, <_K)$ of T = (AA,<) (for k=1,...,n). A global log L of $T_1,\ldots,T_n$ is a partially ordered set (BB,<<) such that 1) BB = $BB_1$ U ... U $BB_n$, and 2) for every A and B in $T_K$, if A $<_K$ B, then A << B in L, and for Send(X,M) in $T_K$ and Receive(M,Y) in $T_J$, Send(X,M) << Receive(M,Y). □

[Definition] A global log L of $T_1,\ldots,T_n$ is said to be serializable to T iff all meaningful ordering relations in T are held in L. □

The serializability gives the criteria of correctness of parallel executions of subtransactions, since T is assumed to be correct.

[Proposition] Any global log of decomposed subtransactions are serializable.
[Proof] Let T be a simple transaction, $T_1,\ldots,T_n$ be decomposed subtransactions of T, and L a global log (BB,<<) of T. According to the decomposition procedure, the conflict and intrasite flow relations in T are held in subtransactions. For an intersite flow relation A => B, subtransaction $T_K$ has A => Send(X,M) and $T_J$ has Receive(M,X) => B. Since Send(X,M) << Receive(M,X) in L, A => Send(X,M) << Receive(M,Y) => B. Hence, A << B in L.■

Hence, our decomposition is correct. That is, the result of L is the same as the serial execution of T.

## 6. CONCLUDING REMARKS

In this paper, we discussed how to decompose Prolog transactions to subtransactions in order to concurrently execute the subtransactions. First, we defined two types of meaningful sequences, conflict and flow ones. The

conflict relations represent read-write and write-write conflict relations in the conventional concurrency control[BERN,GRAY]. The flow relations represent the input-output relations among the atoms. In the resolution in Prolog, a variable is instantiated only once. This means a single assignment property of data flow languages [ACKE]. By this property, it is easy to find the conflict and flow relations. We define two kinds of deadlocks, i.e. forward and backward ones. W showed the method to decompose the transactions into deadlock-free subtransactions on send-receive relation for the forward deadlock. For the backward deadlock, a control mechanism which detects the deadlock and break it by selecting a subtransaction to be further backtracked.

## REFERENCES


[ACKE] Ackerman, W.B. and Dennis, J.B., "VAL - Value-Oriented Algorithmic Language," LCS/TR-218, MIT, 1979.

[BERN] Bernstein, P.A. and Goodman, N., "Concurrency Control in Distributed Database Systems," ACM Computing Surveys, Vol.13, No.2, 1981.

[CLAR] Clark, K.L. and Gregory, S., "PARLOG: Parallel Programming in Logic," Research Report DOC 84/4, Imperial College of Science and Technology, London, 1984.

[CLOC] Clocksin, W.F. and Mellish, C.S., "Programming in Prolog," Springer-Verlag, 1984.

[CODD] Codd, E.F., "A Relational Model of Data for Large Shared Data Banks, " CACM, Vol.13, No.6, 1970.

[CODA] Codasyl, "Report of the CODASYL Data Definition Language Committee," Journal of Development, 1973.

[DATE] Date, C.J., " Introduction to Database Systems," Addison-Wesley, 1983.

[GRAY] Gray, J., "The Notions of Consistency and Predicate Locks in a Database System," CACM, Vol.19, No.11, 1976.

[HEVN] Hevner,A. and Yao,S.B., "Query Processing on a Distributed Databases," Proc. of the 3rd Berkeley Workshop, 1978, pp.91-107.

[KOWA] Kowalski, R.A., "Logic for Problem Solving," Elsevier North Holland, 1979.

[LAND] Landers, T. and Rosenberg, R.L., "An Overview of Multibase," North-Holland, 1982.

[MOSS] Moss, J.E.B., "Nested Transactions," MIT Press, 1985.

[NAIS] Naish, L., Thom, J.A., and Ramamohanarao, "Concurrent Database Updates in Prolog," Proc. of the Fourth International Conf. on Logic Programming, 1987, pp.178-195.

[OLLE] Olle, T., "The CODASYL Approach to Data Base Management," John Wiley and Sons, 1978.

[SHAP] Shapiro, E., "Concurrent Prolog: A progress Report," Foundation of Artificial Intelligence, Springer-Verlag, 1986.

[TAK87a] Takizawa, M., Ito, H., and Moriya, K., "Logic Interface System on the Navigational Database System," Lecture Notes in Computer Science, No.264, 1987, pp.70-80.

[TAK87b] Takizawa, M., "Highly Reliable Broadcast Communication Protocol," Proc. of IEEE COMPSAC, Tokyo, 1987, pp.731-740.

[TAK87c] Takizawa, M., "Cluster Control Protocol for Highly Reliable Broad cast Communication," Proc. of the IFIP Conf. on Distributed Processing, Amsterdam, 1987.

[TAK88a] Takizawa,M., Katsumata,M., and Nagahora,S., "Access Procedure to Minimize Redundant Refutations," Proc. of the logic Programming Conf., ICOT, 1988, pp.187-196.

[TAK88b] Takizawa, M. and Miyajima, K., "Concurrent Execution of Prolog Transaction," to appear in Lecture Notes in Computer Science, Springer-Verlag, 1988.

[UEDA] Ueda, K., "Guarded Horn Clauses," Lecture Notes in Computer Science, 221, Springer-Verlag, 1986.

[ULLM] Ullman, D., "Implementation of Logical Query Language for Databases," ACM TODS, Vol.10, No.3, 1985, pp.289-321.