Incremental Protocol Verification Using Deductive Database Systems*

I-En Liao and Ming T. Liu

Department of Computer and Information Science
The Ohio State University
Columbus, Ohio 43210-1277

ABSTRACT
Reachability analysis has been widely used in validating and verifying communication protocols; however, it is incapable of expressing the verification procedure elegantly, and it also suffers from the state space explosion problem. A new approach using the relational database system due to Lai and Lee [6, 7] does alleviate the above problems. In this paper, we first incorporate the predicate logic into the relational algebraic approach so that the recursive definitions are permitted and different functional properties of a protocol can be proved by providing users with a uniform query interface. Next, we apply our approach to incremental verification, which allows for a complex protocol to be verified incrementally. Our scheme is intended to cut down the verification time by saving the explored global-state transitions in the database so that whenever the specification is modified and the verification needs to be performed again, only a small portion of the database will be updated without regenerating all the global states from the initial state.

1. Introduction
As the use of distributed computing systems becomes more and more popular, various protocols for different network layers have been proposed. To ensure that a protocol will work properly, we must prove the correctness of the protocol. The ways we used to guarantee the correctness of a protocol are to validate the protocol for freedom of syntactical errors (also called general properties) and to verify it for conformance to its functional properties [12]. Techniques used for the former are called protocol validation; and those for the latter protocol verification. In this paper, we will use the word "verification" to include the validation of a protocol, but not vice versa. Most of the techniques proposed for protocol validation and verification are based on reachability analysis [8], due to its ease of mechanization. However, reachability analysis is incapable of expressing the verification procedure elegantly, and it also suffers from the state space explosion problem when the protocol to be verified is complex.

A new approach using relational database [6, 7] can alleviate those problems by expressing the validation procedure in terms of relational algebra and by utilizing the optimization mechanisms embedded in the database management system. This approach does provide some nice results in validating the general properties of a protocol; however, it's still difficult to handle some complicated queries such as recursions. Moreover, Lai and Lee did not apply the relational-algebraic approach to verify the functional properties of a protocol. Therefore, in this paper we incorporate logical reasoning into the relational algebraic approach to construct a deductive relational database for protocol verification.

Based on the deductive relational database, we then apply it to incremental verification for verifying protocols incrementally. In a verification cycle (see Figure 1), saving the explored global states in the database is beneficial. In cases where the specification needs to be modified to fix some errors, verifying the modified protocol involves only adding or deleting a small portion of the database without regenerating all the global states from scratch. This is advantageous, especially for the verification of a complicated protocol where a complete verification takes hours to finish.

Figure 1: Verification Process.

We organize this paper as follows. In Section 2, we describe the relational algebraic approach proposed by Lai and Lee [6, 7], and discuss the merits of their approach. Section 3 shows our deductive method and its application to protocol verification. Section 4 describes the incremental verification technique. Related work and non-quantitative comparison with our approach are briefly discussed in Section 5. Finally, we present our conclusions in Section 6.
2. Relational Algebraic Approach

In the relational algebraic approach, a communicating automaton is first represented by a transmission relation and a reception relation. Then relational algebra is utilized to derive the global-state transitions of the system. After all the global-state transitions have been generated, some syntactical errors such as deadlock, incomplete specification, and non-executable code can be easily identified because they can be formulated in terms of relational algebra.

2.1 Tabular Representation of Protocol

A protocol can be modeled as a pair of communicating entities, say process A and process B, which in turn are represented by finite-state machines [1]. We assume that the channel connecting the two entities is an error-free, FIFO channel. Later in Section 4, we will relax this assumption, where a demon is introduced to model transmission errors. The most commonly used representation for a finite-state machine is the directed graph model, in which the node represents the state of the entity and the arc indicates the message that triggers the entity state transition. Following the notions used in [1], a minus sign denotes a message transmission, and a plus sign means a message reception.

Figure 2 is a simple USER-SERVER protocol taken from [1], which will be used throughout this paper. In this protocol, the USER while in its initial state READY can initiate a request by sending REQ to the SERVER, and then enters the WAIT state. After receiving request message REQ, the SERVER enters state SERVICE. Upon completion of its service, the SERVER sends a response message DONE to the USER, and then changes to state IDLE. While in state IDLE, the SERVER can detect a fault in itself. If so, it informs the USER by sending message ALARM. When the USER receives message ALARM, it responds by sending message ACK to direct the SERVER back to its IDLE state upon receipt of message ACK.

In the relational algebraic approach, an entity is represented by two relations: the transmission relation and the reception relation. Each relation has three attributes, namely the current state (A or B), the trigger (Ma or Mb), and the next state (A' or B'). Figure 3 shows the corresponding relations for the USER-SERVER protocol.

In order to prove the correctness of a protocol, all the global states must be generated. Here, a global state consists of four components: (1) the state of process A (denoted as A or A') in Figure 4, (2) the state of process B (denoted as B or B') in Figure 4, (3) the message in the A-to-B channel (denoted as X or X' in Figure 4), and (4) the message in the B-to-A channel (denoted as Y or Y' in Figure 4). We also use \([A, X, Y, B]\) to represent a global state. Assuming the channel capacity is one, Figure 4 shows the global-state transition diagram for the USER-SERVER protocol. The corresponding global-state transition relation is shown in Figure 5.

Each tuple of the global-state transition relation corresponds to a global state transition. In general, the global-state transition relation of a 2-entity protocol is a relation on a set of \((4n + 6)\) attributes, namely \(A, X_1, \ldots, X_n, Y_1, \ldots, Y_n, B, M_A, M_B, A', X'_1, \ldots, X'_n, Y'_1, \ldots, Y'_n, B'\), where \(n\) is the channel capacity. The attributes \(X_i\) and \(Y_i\) represent the \(i\)th and \(j\)th messages in transit to be received by process B and A, respectively. If the channel capacity \(n\) is greater than one, we may still use \(X, Y, X',\) and \(Y'\) to represent the whole contents of the channels when there is no confusion arising.

Figure 4: Global State Transition Diagram of the USER-SERVER Protocol.
Given the transmission and reception relations of the participating processes and the initial global state, the main task in the relational algebraic approach is to generate the global-state transition relations. Lai and Lee [6, 7] introduced six operations in relational algebra to derive an algorithm for that purpose. These six operations are PROJECTION(\(\pi\)), SELECTION(\(\sigma\)), RENAMING(\(\rho\)), REPLACEMENT(\(\eta\)), JOIN(\(\rho\)), and THETA-JOIN(\(\theta\)). They are defined as follows.

A relation \(R\) on a set of attributes \(\Omega = \{A_1, \ldots, A_n\}\) is denoted by \(R[\Omega]\) or \(R[A_1, \ldots, A_n]\). Let \(t\) be a tuple in \(R[\Omega]\). The components of \(t\) corresponding to the set of attributes \(X\), where \(X\) is a subset of \(\Omega\), is denoted by \(t[X]\). If \(a_i\) is a constant from the domain of \(A_i\), then \(t[A_i < a_1; A_2, \ldots; A_n:] = t[A_i > a_1; A_2, \ldots; A_n] > \) is a relation representing a constant tuple over \(\Omega\). We now introduce the definitions of those six relational operations.

1. **Projection:** The projection of \(R\) on \(X\) is the relation \(\pi_X(R) = \{t[X] | t \in R\}\).

2. **Selection:** The selection of the tuples of \(R\) which satisfy the formula \(f\) is the relation \(\sigma_f(R) = \{t | t \text{ satisfies formula } f\}, t \in R\).

3. **Renaming:** The relation \(R[\Omega]\) with attribute \(A \in \Omega\) renamed to \(B \notin \Omega\) is the relation \(\rho_{\Omega \to \{A \to B\}}(R) = \{t' | t'[\Omega \to B] = t[\Omega \to A], t'[B] = t[A], t \in R\}\)
defined on the set \((\Omega - A)B\).

4. **Replacement:** The relation \(R[\Omega]\) with column \(A\) replaced by column \(B\) is the relation \(\eta_{\Omega \to \{A \to B\}}(R) = \{t' | t'[\Omega \to B] = t[\Omega \to A], t'[B] = t[A], t \in R\}\)
defined on \(\Omega \cup \{B\}\), where \(A \in \Omega\).

5. **Join:** The join of \(R[XZ]\) and \(S[YZ]\), denoted by \(R[XZ] \Join S[YZ]\), is the relation \(R[XZ] \Join S[YZ] = \{t | t[XZ] \in R, t[YZ] \in S\}\), where \(X, Y, Z\) are disjoint sets of attributes.

6. **Theta-join:** Let \(R[X]\) and \(S[Y]\) be two relations such that \(X \cap Y = \emptyset\). Then \(R[X] \Join \pi_f f[S] = \text{the relation}

\[T[XY] = \{t | t \text{ satisfies formula } f \text{ and } t[X] \in R, t[Y] \in S\}\]

In order to find the global-state transition relation, the following four more operations were defined in [6, 7]:

1. The transmission operation \(\oplus\) for process \(A\): Given a set of global states \(S\) and the transmission relation \(T_a\) of process \(A\), the operation \(S \oplus T_a\) will yield the set of all possible global-state transitions from \(S\) obtained by transmitting one message from process \(A\). It is defined by the following procedure:

   - step 1: \(H \leftarrow S[x]; \)
   - step 2: \(H \leftarrow \eta_{X \to Y \to B} \delta_{X \to Y \to B}(H)|x| < 0; M_0 >;\)
   - step 3: \(Q \leftarrow \emptyset;\)
   - step 4: \(Q \leftarrow \{H \leftarrow H - R;\}
   - step 5: \(S \oplus T_a \leftarrow Q;\)

2. The reception operation \(\ominus\) for process \(A\): This operation, written as \(S \ominus R_a\), will generate all the possible global-state transitions from \(S\) obtained by receiving a message from channel \(B\)-to-\(A\). It is defined by the following procedure:

   - step 1: \(H \leftarrow S[Y] = M_a; R_a;\)
   - step 2: \(H \leftarrow \eta_{X \to Y \to B} \delta_{X \to Y \to B}(H);\)
   - step 3: \(S \ominus R_a \leftarrow H|X| < 0; Y_a >; M_0 >;\)

Similarly, we can define the transmission and reception operations for process \(B\). Using these operations, Lai and Lee derived an elegant algorithm to generate the global-state transition relation. After the global-state transition relation is found, some design errors such as deadlock, incomplete specification, non-executable interaction, and state ambiguity can be formulated nicely in terms of relational algebra. For example, the set of deadlock states is given by

\[D = \delta_{X \to Y \to B} \delta_{Y \to B} \delta_{X \to Y \to B}(G) - \pi_{X \to Y \to B}(G)\]

where \(G = \text{the global-state transition relation}\).

In the above formula, the first term indicates all the possible global states after the message trigger. The second term indicates all the possible global states before the message trigger. The difference of these two terms represents all the possible global states that a system can enter but can't leave.

Due to the use of the well-developed relational database theory, the relational algebraic approach is easy to implement and maintain. Another appealing feature of this approach is its capability to express the syntactical errors of a protocol in terms of relational algebra.
3. Deductive Approach

In the previous section, we have seen how relational algebra can be applied to protocol validation. Although the results are quite neat, there are still some limitations in that approach. First, all the global states must be generated in order to complete the validation. This means an exhaustive state exploration scheme was adopted, and that could be a problem in cases where the complete state exploration is beyond the capacity of a computer system for some complicated protocols. Secondly, relational algebra does not allow recursive definitions in a query. For example, in a family database PARENT whose tuple (A,B) represents the relationship "B is a parent of A," it is very difficult to write a single query based on relational algebra which finds all the ancestors of an individual. Finally, even though the relational algebraic approach can validate the functional properties of protocols such as recovery from transmission errors [Rud 82], thus the lack of capability for handling recursion in relational algebra becomes even severer.

In this section, we introduce the concepts of first-order logic and the deductive database to remedy the two aforementioned problems.

3.1 First-Order Logic and Deductive Database

One branch of logic we used most often in computer science is propositional logic (or propositional calculus), which is the logic of propositions connected by \( \neg \) (NOT), \( \land \) (AND), \( \lor \) (OR), \( \rightarrow \) (IF ... THEN), and \( \leftrightarrow \) (IF AND ONLY IF). An application of this logic is the circuit design in computer hardware in which truth tables are used to determine the truth or falsity of formulas. Although propositional logic is very useful in some applications, it is not powerful enough to handle many types of logical reasoning. Therefore, three more logical notions (called terms, predicates, and quantifiers) are added to enrich propositional logic. The extended logic is called first-order logic (or first-order predicate calculus), which allows for more complex logical reasoning.

The use of logic in database design has been studied intensively. A good survey paper in this area can be found in [4]. Basically, a deductive database consists of two components: an extensional database (the tables) and an intensional database (the rules) [5]. The rules are expressed in first-order logic and used to define the intensional database in terms of the extensional database. Since Prolog is a language whose statements are formulas of first-order logic, we thus use a Prolog-like language as the query language for our new deductive approach. As an example of a deductive database, Figure 6 shows the representation of Figure 5 as a deductive relational database. There are four axioms defined in the intensional database. Two of them are used to define predicate symbol SUCC(essor) and the other two for ANCES(essor).

For simplicity, we use Figure 6 as an example to illustrate the syntax of our deductive database. Let \( TRANS \) be a predicate symbol defined on the tuples of the global-state transition relation. \( TRANS(A,B,C,D,\ldots,P,Q,S,T) \) is true if and only if \( (A,B,C,D,\ldots,P,Q,S,T) \) is a tuple in the transition relation, where the left-out arguments between commas represent the don't-care items. A formula "\( P \land Q \rightarrow R \)" is represented as " \( R:-P, Q, " \) in Prolog. Thus, successor predicate SUCC and ancestor predicate ANCES can be defined recursively in the intensional database. For instance, the definition of \( SUCC \) is composed of two parts. The first part defines \( [P,Q,S,T] \) is a successor of \( [A,B,C,D] \) if \( [P,Q,S,T] \) is the next state of \( [A,B,C,D] \) and \( [P,Q,S,T] \) is not a member of the Set of the Visited States (SVS) along an inference path. \( IN([P,Q,S,T], SVS) \) in the formula is a Boolean function which returns true if \( [P,Q,S,T] \) is a member of SVS, and false otherwise. Notice that Prolog utilizes backtracking to find the solutions to a goal. During the inference process, there is an SVS set associated with each path generated. So if a state has been visited, i.e., it is a member of the SVS, visiting this state again means there exists a loop in the global-state transition graph. Therefore, we must terminate the exploration of this path to avoid an infinite loop in the inference process. The second rule states that \( [P,Q,S,T] \) is a successor of \( [A,B,C,D] \) if \( [W,X,Y,Z] \) is the next state of \( [A,B,C,D] \) and \( [P,Q,S,T] \) is a successor of \( [W,X,Y,Z] \).

For ancestor predicate symbol ANCES, we define it slightly different. The initial global state would be the ultimate ancestor of any state, and any state in the Set of the Visited States (SVS) may not be visited twice. The main reason for these distinctions is because \( SUCC \) is used in the tracing algorithm [13] to verify the functional properties of a protocol, while \( ANCES \) is mainly for finding the paths to a particular state.
An example of the query in this deductive database is given in part (c) of Figure 6. To answer a query, the resolution method of theorem proving [11, 3] is employed.

3.2 Verifying Protocols by Deductive Relational Database

To verify a protocol, we first construct the transmission and reception relations for the participating processes. We then employ the algorithm derived in [6] to generate the global-state transition relation, which also constitutes the extensional database in a deductive database. Finally, the axioms defining SUCC and ANCES, as in part (b) of Figure 6, are added to the database. Some general properties can be validated by relational algebra as explained in Section 2. Now with the recursive and logical reasoning properties of the deductive database, we are able to answer some queries such as the paths leading to a deadlock state.

Finding the paths to a particular state could be very useful in analyzing and debugging a protocol because the designer would be able to understand the interactions between two entities which produce the resulting state. For instance, global state \([1,6,9,5]\) in the USER-SERVER protocol shown in Figure 4 is a deadlock state, and it also has incomplete specification. These errors can be detected by relational algebra, but there is no simple query in relational algebra that can find all the paths to the state \([1,6,9,5]\). The original relational algebraic approach proposed by Lai and Lee relies upon manual checking for finding the paths, while our approach automates this process.

Figure 7 shows the resolution method for finding the paths to state \([1,6,9,5]\). The first column states the goals to be achieved in the resolution process. The second column indicates the rules to be applied for generating subgoals, whereas the last column shows the substitutions used for the variables in the rules, for example, \([A/X]\) means substituting \(A\) for \(X\). The resolution process will not terminate until a conclusion, either success or failure, can be drawn for a subgoal and all the possible inference rules have been tried.

Another nice feature about our approach is its capability of verifying the functional properties of a protocol. To verify that a protocol has certain functional properties, one can prove that given an initial condition, another desirable condition is eventually reached. Tracing [13] is one of the techniques used in proving the functional properties. It works as follows. A trace of all the possible paths starting from the global states satisfying the initial condition is generated. The trace continues until either the desired condition is reached (and at that point the trace is successfully terminated), or until a specified number of iterations is reached without reaching the desired condition (and at that point the protocol has an error). Consider, for example, the work done by Rudin [13] at the IBM Zurich Lab. They demonstrated that the token-ring protocol is robust in the sense that a satisfactory recovery occurs in all cases where a demon introduces transmission errors. There are

![Figure 7: Finding the Paths to a Deadlock State by Resolution.](image-url)
nine property-proving programs used to verify the tokenring protocol. The first program showed that the control flow was correct when the demon made no errors. The other eight programs showed that the recovery was complete in case of transmission errors.

In our deductive approach, we first use predicate symbol TRANS to identify all the global states which satisfy the initial condition. Then apply SUCC to those states to see if the desired condition can be reached in all cases. In essence, we also utilize the tracing method, but our approach provides a uniform query interface to the user. The user needs only to put different conditions in a query in order to verify different properties.

In addition to finding paths and verifying functional properties, the deductive database we constructed in this section can serve as the foundation for our incremental verification method, which is to be explained in the next section.

4. Incremental Verification

The relational algebraic approach is one of the "complete" validation techniques in the sense that "all" the possible global states must be generated in the validation process. In fact, it keeps the following relations in the database:

\[ T_A = \text{transmission relation of process } A, \]
\[ R_A = \text{reception relation of process } A, \]
\[ T_B = \text{transmission relation of process } B, \]
\[ R_B = \text{reception relation of process } B, \]
\[ S = \text{global state relation,} \]
\[ G = \text{global-state transition relation.} \]

Once the above relations of a protocol are saved in the database, the reverification of that protocol is relatively easy and less time-consuming. What we need to do is to identify those transitions affected by the changes of the specification, and then update the database accordingly. In this section, we demonstrate how the changes in the specification can be translated into the updates of the database. Here, a transmission or reception of a message by a process exists a derivation path from a resulting state of \( E \) to \( S \) and we call a reachable state \( S \) of an event \( E \) uniquely determined by \( E \) if there is only one path from a resulting state of \( E \) to \( S \).

4.1 Deletion of an Event Specification

Suppose an event specification, say \( a: A \rightarrow m: M, a': A' \rightarrow \), is to be deleted from process A. Then some of the reachable global states of this event should be eliminated from the database. Because the interactions of protocol entities may be very complex, a global state could have several predecessor states. So one subtle problem of deleting an event specification is how to determine if a state reachable from the event should be deleted or not. We use the following simple criterion to check all the resulting states of the deleted event: if a resulting state has only one predecessor state, then it should be eliminated; otherwise, it is called a contaminated state and should be put into a contaminated set waiting for further examinations. The detailed algorithm is given as follows:

\[
\text{step 1: } G_1 \leftarrow \sigma_{A = m, M = m}(\pi_{A \rightarrow M}(G)) \times G; \\
/* \text{Identify the global-state transitions triggered by this event. */}
\]

\[
\text{step 2: } S_1 \leftarrow \delta_{A \rightarrow X, Y \rightarrow B}(\pi_{A \rightarrow X, Y \rightarrow B}(G_1)) - \{I\}; \\
/* \text{Identify the global states reached by this event in one derivation. The initial global state } I \text{ should be excluded. */}
\]

\[
\text{step 3: } D \leftarrow \text{Singleparent}(S_1); \\
C \leftarrow S_1 - D; \\
H \leftarrow D; \\
/* \text{The function Singleparent will return all the elements of } S_1 \text{ which have only one predecessor state. } \\
D \text{ is the set of states known to be deleted. } C \text{ is the set of contaminated states. */}
\]

\[
\text{step 4: DO UNTIL } H = \emptyset; \\
/* H \text{ is the set of states to be deleted */}
\]

\[
N \leftarrow (\sigma_{X \rightarrow a}(H) \otimes T_A) \cup (\sigma_{X \rightarrow a}(H) \otimes T_B) \\
H \leftarrow (H \oplus R_A) \cup (H \oplus R_B); \\
G_1 \leftarrow G_1 \cup N; \\
S_2 \leftarrow \delta_{A \rightarrow X, Y \rightarrow B}(\pi_{A \rightarrow X, Y \rightarrow B}(N)) - S_1 - \{I\}; \\
S_1 \leftarrow S_1 - S_2; \\
H \leftarrow \text{Singleparent}(S_1); \\
D \leftarrow D \cup H; \\
C \leftarrow C \cup (S_2 - H); \\
\]

\[
\text{step 5: } G \leftarrow G - G_1; \\
/* \text{Update the global-state transition relation */}
\]

\[
S \leftarrow S - D; \\
/* \text{Update the global state relation */}
\]

\[
\text{step 6: FOR each } s \in C \\
/* \text{Check if } I \text{ is an ancestor of } s */
\]

\[
\text{IF ANCES}(s, I) \\
\quad \text{THEN } C \leftarrow C - \{s\}; \\
\]

\[
\text{step 7: IF } C \neq \emptyset \\
\quad \text{THEN } D \leftarrow C; \\
\quad H \leftarrow C; \\
\quad S_1 \leftarrow C;
\]

221
step 8: IF m > 0
   /* Assume 0 is never used as a message */
   THEN T_a <- T_a - < a: A m : M_a a' : A' >
   /* Update transmission relation */
   ELSE R_a <- R_a - < a: A m : M_a a' : A' >;
   /* Update reception relation */
   G <- G ∪ N;
   H <- δ_{xyb} = -axyb(b(σ_{x = y}(N)) - S);
   S <- H ∪ S;

This algorithm is almost the same as the one used in [6] for generating the global-state transition relation. But our algorithm does not recompute all of the global states; instead, it only explores the states which are possible to trigger transitions because of the addition of the new event specification. Also notice that the transmission and reception relations are updated before the updates of G and S. The addition of an event specification to process B can be defined in the same way. If the modification of a protocol specification involves both deletion and addition of events, then the deletion will be performed before the addition for the sake of efficiency.

In this section, we make use of the relational database and the deductive relational database to define four algorithms for reverification of a protocol. Since these algorithms only update the database for the transitions affected by the changes in the specification, they indeed provide an efficient technique for incrementally recalculating the global-state transitions.

5. Related Work

The protocol synthesizer proposed by Brand and Za- firopulo [2] is a protocol construction tool whose function can be specified as follows: Given N finite state machines representing a partially constructed protocol, and given a new transmission, find how the protocol must be extended. This approach can be considered as an incremental specification technique in the sense that the specification of a protocol can be completed gradually by interacting with the synthesizer during the specification process. Therefore, it should be run in an interactive environment; otherwise, the global state tree needs to be regenerated in the batch systems. But our incremental verification approach will not have that problem, because the explored global states are saved in the database.

The use of Prolog in the protocol verification have been reported in [15, 14, 16]. The so called incremental development in their papers was meant to apply the Prolog to every step of a protocol development: protocol service specification and validation, protocol verification, generating protocol test sequences, and protocol simulation. Among other things, Prolog has a serious drawback. That is, it is not feasible for large databases because Prolog is usually implemented without convenient access to external databases [5]. In contrast with Prolog, our approach is an attempt to couple Prolog with the data manipulation operations in relational algebra.
6. Conclusion

The use of reachability analysis in protocol verification has been restricted by the state space explosion problem. Most of the validation software developed, for example the TTG validation system at OSU [10], use linked lists as the underlying data structures for the global states generated during the reachability analysis, in which no storage optimization is attempted. Hence, the validation system will run out of memory space even for a simple protocol [9]. This problem can be alleviated by the relational algebraic approach which takes advantages of the well-developed database management system to handle the storage and query optimizations. Another notable feature of the relational algebraic approach is the ability to express validation procedure elegantly by using relational algebra. The design errors which can be expressed in terms of relational algebra include deadlock, incomplete specification, non-executable interaction, state ambiguity, and temporal blocking loops.

However, the relational algebraic approach cannot easily display the sequence of transitions along a path to a particular state because it doesn't allow recursive definitions in a query. Therefore, in this paper we propose a deductive approach which incorporates first-order logic into the relational database to remedy the problems incurred in the relational algebraic approach. Our method not only allows recursive definitions for more complex logical reasoning, but also provides a uniform query interface to the users for verifying functional properties of a protocol. Most importantly, with the deductibility of the deductive database, we are able to define algorithms for the incremental verification method, which speeds up the verification process by verifying protocols without generating the global states from scratch.

Because both the relational algebraic and deductive approaches require all the system transitions be generated before the verification procedure can proceed, it is conceivable that even with a database system capable of handling a huge amount of data, the response time could be intolerable for a complex protocol. Currently, we are evaluating the performance of our approach, and investigating the possibility of adding heuristic algorithms into the incremental method so that incomplete verification can be performed and some design errors can be detected efficiently without generating all global-state transitions.

References