Proceedings of the 23rd Annual Virginia Computer User's Conference, 47–51.
Blacksburg, VA: Department of Computer Science, Virginia Tech.

Protocol Verification using Relational Algebra

Quynh-Anh Nguyen[†] Marcus A. Maloof[‡]
qnguyen@mason1.gmu.edu maloof@aic.gmu.edu

[†]Department of Computer Science
[‡]Center for Artificial Intelligence
George Mason University
Fairfax, VA 22030-4444

Abstract

A communication protocol is a set of rules governing data exchange between two communicating processes. Specification of these protocols is critical for ensuring an error-free framework for implementing and executing communication systems. The purpose of protocol verification is to test protocol specifications and detect any operative errors. Due to the important roles communication protocols play in today's world of information processing, it is vital that they are specified and verified in the most effective and efficient means. Finite-state machines provide a natural model for specifying protocols. When represented using transition tables, which correspond directly to a relational data model, the algorithms necessary to verify protocols can be specified in relational algebra. This paper describes a system that represents protocols as relations and implements protocol verification algorithms using relational algebra in a logic programming framework. Using this system, experiments were conducted on several synthetic communication protocols to determine if the protocols were well-behaved.

1 Introduction

A *communication protocol* is a set of rules governing data exchange between two communicating processes. It is critical that these protocols are specified correctly in order to maintain an error-free framework for implementing and executing communication systems. The purpose of protocol verification is to test protocol specifications and detect any operative errors such as deadlocks, incomplete specifications, and nonexecutable interactions. Due to the important roles communication protocols play in today's world of information processing, it is vital that they are specified and verified in the most effective and efficient means.

One approach used in specifying protocols is to model individual processes as finite-state machines (Lee and Lai 1988). Transmitting messages to or receiving messages from other processes corresponds to state changes in the finite state machine. Based on this style of specification, Lee and Lai introduce an innovative technique for protocol verification by encoding protocols using a relational data model, introduced and developed by Codd (1990), and expressing the the protocol verification algorithm using relational algebra.

Using Lee and Lai's verification algorithms, we implemented a protocol verification system in Prolog, much like Liao and Liu (1989). The system was used to verify several synthetic protocols, including some supplied by Lee and Lai. This paper discusses the experiments and findings using Lee and Lai's protocol verification algorithms and criteria for well-behaved protocols.

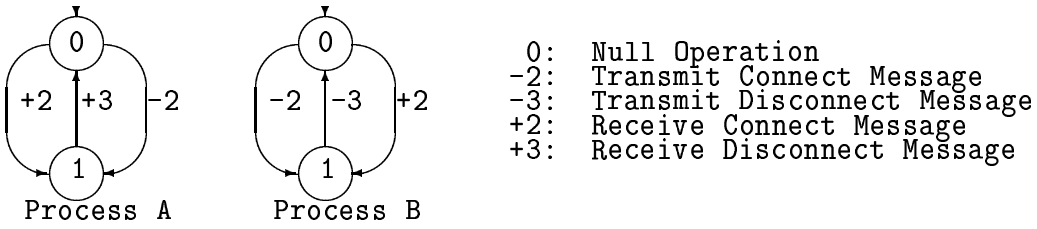

Figure 1: Transition diagrams for a simple protocol (Lee and Lai 1988).

2 Background

This section provides a brief background on the topics pertinent to Lee and Lai's approach and our implementation. The first sub-section outlines finite-state machines, the formalism used to represent communication protocols. The second discusses the relational data model and relational algebra, which is used to represent finite-state machines and and express protocol verification algorithms. Finally, the third sub-section briefly discusses Prolog, the language in which we implemented the approach.

2.1 Finite-State Machines

Briefly, Hopcroft and Ullman (1979) define a finite-state machine (FSM) as a 5-tuple:

$$(Q, \Sigma, \delta, q_0, F),$$

where

Q is a finite set of *states*,
Σ is a finite *input alphabet*,
$q_0 \in Q$ is the *initial state*,
$F \subseteq Q$ is the set of *final* states, and
δ is a *transition function* such that $Q \times \Sigma \rightarrow Q$. In other words, for a given state and letter from the alphabet, δ yields the new state of the machine.

To specify protocols, each symbol in the input alphabet is prefixed with either a minus sign, signifying a transmission signal, or a plus sign, signifying a reception signal. Since each finite-state machine has send and receive behaviors, constituting a communication process, we can model a protocol using two finite-state machines. Figure 1 shows the state-transition diagram for a simple connection-establishment protocol. The transmission and reception behaviors of the protocol can be summarized in tabular form, as shown in Figure 2.

2.2 Relational Algebra

In a relational data model, originally introduced by Codd in 1970, data is organized into a tabular format, much like the transition tables depicted in Figure 2. These tables are formally referred to as *relations*. Each column in the table is an *attribute*, taking values from

q_a	σ_a	q'_a
0	-2	1

Process A Transmission Table

q_a	σ_a	q'_a
0	+2	1
1	+3	0

Process A Reception Table

q_b	σ_b	q'_b
0	-2	1
1	-3	0

Process B Transmission Table

q_b	σ_b	q'_b
0	+2	1

Process B Reception Table

Figure 2: Protocol transition tables (Lee and Lai 1988).

q_a	σ_a	q'_a
1	+3	0

$\sigma_{q_a=1}(R_a)$

q_a	q'_a
0	1
1	0

$\pi_{q_a q'_a}(R_a)$

p	σ_a	q'_a
0	+2	1
1	+3	0

$\delta_{p \leftarrow q_a}(R_a)$

Figure 3: Example relational operations on Process A's Reception Table (R_a).

a *domain* that specifies all possible values an attribute can take. Each row, or element in the relation is called a *tuple.*

Referring to Figure 2, if we take Process A's reception table as an example, the table itself is a relation having attributes q_a, σ_a, and q'_a. The domain of q_a and q'_a is simply 0 or 1, since both represent FSM states and the Process A FSM is a two state machine. σ_a, on the other hand, has the domain $\{0, +2, +3\}$, as defined in Figure 1.

Several operations exist for manipulating relations. In addition to the usual set operations such as union, intersection, Cartesian product, and set difference; other relational operators include *select*, *project*, and *rename*. This list is not exhaustive, but are the pertinent operators Lee and Lai used. See Ullman (1988) for a more extensive list.

The select operator (denoted using σ) yields a new relation by finding tuples with specific attribute values. The project operator (denoted using π) is used to create a new relation by isolating one or more specified columns of an existing relation. Finally, the rename operator (denoted using δ) results in a new relation by changing the name of one or more attributes.

Figure 3 demonstrates these operations on Process A's Reception Table (denoted using R_a). $\sigma_{q_a=1}(R_a)$ creates a relation having tuples such that the values of the attribute q_a are 1. $\pi_{q_a q'_a}(R_a)$ produces a relation having attributes q_a and q'_a. Finally, $\delta_{p \leftarrow q_a}(R_a)$ returns a relation in which the attribute q_a is renamed to p.

2.3 Prolog

Prolog is a logic programming language based on a subset of first-order predicate logic. Specifically, Prolog statements are *Horn clauses*, which are clauses in disjunctive normal

form and having only one positive literal. Given that p_i is a literal, we have the clause:

$$p_1 \vee \neg p_2 \vee \neg p_3 \vee \cdots \vee \neg p_n$$

which we can easily re-write[1] as

$$p_1 \subset p_2 \wedge p_3 \wedge \cdots \wedge p_n$$

In Prolog syntax, we would write this clause as

```
p1 :- p2, p3,...,pn.
```

which is read "p1 is true only if p2 is true and p3 is true and ... and p_n is true." Covington et al. (1986) provide more details on the logical basis for Prolog.

Regardless of the formal underpinnings of Prolog, since migrating to the United States from Europe, Prolog has gained popularity. Through this popularity, researchers have discovered Prolog as an excellent language for prototyping and developing relational database systems (Gray and Lucas 1988; Kazic et al. 1989). Additionally, there is a natural translation between relational algebraic statements, also having their foundations in formal logic, and Prolog clauses (Mouta et. al 1988; Ullman 1988).

Prolog *facts* are used to represent the tuples in protocol transition tables. For example, the Process A Reception Table in Figure 2 can be represented using the following facts:

```
ra(0, 2, 1).
ra(1, 3, 0).
```

The remaining tables can be represented similarly.

Prolog *clauses* can be used to formulate relational algebraic queries. Referring to Figure 3,

$$\sigma_{q_a=1}(R_a)$$

can be written in Prolog as,

```
select :-
   ra(1, Sigma, Qprime),
   printlist([1, Sigma, Qprime]),
   fail.
select.
```

where `printlist` simply prints list elements. Alternatively instead of printing it, we could have either modified the existing relation or created a new relation. Likewise,

$$\pi_{q_a q'_a}(R_a)$$

can be written in Prolog as,

```
project :-
   ra(Q, _, Qprime),
   printlist([Q, Qprime]),
   fail.
project.
```

[1] Using Associativity, DeMorgan's, Tautology, and Transposition (see Mates 1972).

3 Approach

To implement their approach for protocol verification using relational algebra, Lee and Lai (1988) make two assumptions that decreases computation. The first is that no process can send a message until a previously-sent message has been received. The second is that no two processes will send each other a message at the same time. A system meeting these conditions is a *steady system*. A system having no messages in the communication channel is a *stable system*. Therefore, the communication process involves transitioning between steady and stable states.

Based on these assumptions and given a protocol represented as a FSM in a tabular format, Lee and Lai define algorithms implemented in relational algebra that compute all possible global states (which is also a FSM) that the message system can enter. A global state is a unique combination of valid starting and ending states for Processes A and B and their respective transmission or reception messages. Using this system transition table, additional relational operations analyze the protocol's behavior and establish its reliability. The general outline of the algorithm is as follows:

```
generate_system_transition_table :-
    compute_steady_transitions,
    compute_reachable_states,
    compute_reachable_transitions,
    compute_global_steady_transitions,
    compute_global_states,
    compute_global_transitions.
```

Lee and Lai define a protocol as *well-behaved* if it is free of *deadlock*, is *completely specified*, and has *no nonexecutable interactions*. Each condition can be determined by executing relational algebraic queries on the system transition table. Deadlocks occur when two or more processes are expecting some type of transition, be it a transmission or a reception of a message, which never occurs. Thus, the system will be stuck in this state once it is reached. In the context of a system transition diagram, a deadlock state is one in which arcs enter a state, but no arcs leave the state.

An incomplete specification arises when a transmittal message is specified, but without a corresponding receiving message. Therefore, even if the communication system is not deadlocked, some messages will not be received.

Nonexecutable interactions arise when transitions exist but cannot occur because of an inherent flaw in the protocol. Typically these result from design flaws and require correction. These transitions will appear in the original specification of the protocol, but will not have corresponding transitions in the system transition diagram.

Five protocols were verified with our Prolog verification system. These protocols were analyzed for deadlock states, states in which nonexecutable interactions occur, and states having incomplete specifications. Table 1 summarizes our results. As mentioned before, the connection establishment and the two-process interaction protocols come from Lee and Lai (1988). The remaining protocols are are from Frieder (1992).

Our implementation runs on a DEC Vax 5900 and consists of approximately 600 lines

Protocol	Deadlock States	Incomplete Specifications	Nonexecutable Interactions	Well-behaved
connect-disconnect	Yes	Yes	No	No
two-process interaction	Yes	Yes	Yes	No
modified bi-sync	No	Yes	Yes	No
d16	No	Yes	No	No
d23	No	Yes	No	No

Table 1: Verification results for seven protocols.

of Prolog code. Appendix A lists the verification results for Lee and Lai's simple connect-disconnect protocol.

4 Conclusions

By presenting an algorithm based on the concepts of relational algebra, Lee and Lai introduce a useful technique for verifying communication protocols. Verification systems insure the correctness of the specification of communication protocols. To provide this total assurance, however, the verification system must itself be proven correct. Relational algebra is theoretically formulated and algorithms developed using these models can be formally proven correct.

As serviceable as Lee and Lai's algorithm proved to be, it faces many problems that severely limit its usefulness. Liao and Liu (1989) add the ability to handle recursion in addition to implementing the verification system in Prolog. Frieder (1992) extends Lee and Lai's algorithm to a parallel implementation to verify more complex protocols and achieve greater response time. As a final limitation, Lee and Lai's algorithms only handle two communicating processes.

Nonetheless, these algorithms proved sufficient for verifying the protocols discussed in this paper. Namely, we verified a simple connection-establishment, a two-process interaction, a modified bi-synchronization, d16, and d23 protocols.

Acknowledgements

This research was conducted at George Mason University and supported by an REU (Research Experience for Undergraduates) Supplement to NSF Grant No. CCR–9109804. The authors recognize and greatly appreciate the support of Ophir Frieder. Thanks also goes to Rick Govoni and the anonymous reviewers who provided meaningful comments.

References

Aho, A. V., and Ullman, J. D. (1992) *Foundations of computer science.* New York, NY: W. H. Freeman.

Codd, E. F. (1990) *The relational model for database management.* New York, NY: Addison-Wesley.

Frieder, O. (1982) A parallel database-driven protocol verification system prototype. *Software — Practice and Experience* 22.3 (March) 245–264.

Gray, P. M. D., and Lucas, R. J, eds. (1988) *Prolog and databases: implementations and new directions.* Chichester, West Sussex: Ellis Horwood Limited.

Hopcroft, J. E., and Ullman, J. D. (1979) *Introduction to automata theory, languages, and computation.* New York, NY: Addison-Wesley.

Kazic, T.; Lusk, E.; Olson, R.; Overbeek, R.; and Tuecke, S. (1989) Prototyping databases in Prolog. In Sterling, L., ed., *The Practice of Prolog*, 1–29. Cambridge, MA: MIT Press.

Lee, T. T., and Lai, M. (1988) A relational algebraic approach to protocol verification. *IEEE Transactions on Software Engineering* 14.2 (February) 184–193.

Liao, I., and Liu, M. T. (1989) Incremental protocol verification using deductive database systems. *Proceedings of the 5th International Conference on Data Engineering*, 216–223.

Mates, B (1972) *Elementary logic.* New York, NY: Oxford University Press.

Mouta, F.; Williams, M. H.; and Neves, J. M. (1988) Implementing query languages in Prolog. In Gray, P. M. D., and Lucas, R. J, eds., *Prolog and Databases: Implementations and New Directions*, 13–21. Chichester, West Sussex: Ellis Horwood Limited.

Ullman, J. D. (1988) *Principles of database and knowledge-base systems.* Rockville, MD: Computer Science Press.

About the Authors

Quynh-Anh (Mimi) Nguyen is a sophomore studying Electrical Engineering at George Mason University. Her research interests are virtual reality and its use in unmanned space missions. Mimi can be reached by mail at 12709 Coronation Road; Herndon, VA 22071; or by email at qnguyen@mason1.gmu.edu.

Marcus A. Maloof is a doctoral student in the Center for Artificial Intelligence at George Mason University. He received his Bachelor of Science in Computer Science in 1989 and his Master of Science in Artificial Intelligence in 1992, both from the University of Georgia. His research interests include applying machine learning techniques to problems in machine vision, logic-based reasoning, and formal models of time. He is a member of IEEE and AAAI. Mark can be reached by mail at Center for Artificial Intelligence; 422 Science & Tech II; George Mason University; Fairfax, VA 22030-4444; or by email at maloof@aic.gmu.edu.

A Connection-establishment Protocol

q_a	σ_a	q'_a
0	-2	1

Process A Transmission Table

q_b	σ_b	q'_b
0	+2	1

Process B Reception Table

q_b	σ_b	q'_b
0	-2	1
1	-3	0

Process B Transmission Table

q_a	σ_a	q'_a
0	+2	1
1	+3	0

Process A Reception Table

q_a	q_b	X	Y
1	1	2	2

Deadlock States

q_a	q_b	X	Y
1	0	0	2
1	1	0	0
0	0	0	0
1	1	2	2
0	1	2	0
1	0	3	0

Incomplete Specification States