PROTOCOL SPECIFICATION, TESTING, AND VERIFICATION, C. Sunshine (ed.)
North-Holland Publishing Company

TOWARDS THE UNIFICATION OF THE FUNCTIONAL AND PERFORMANCE ANALYSIS OF PROTOCOLS OR, IS THE ALTERNATING-BIT PROTOCOL REALLY CORRECT?*

Y. YEMINI & J.F. KUROSE
COMPUTER SCIENCE DEPARTMENT
COLUMBIA UNIVERSITY,
NY NY 10027

ABSTRACT

In the past 15 years the alternating-bit protocol has been perhaps the most widely verified protocol, the benchmark of protocol verification techniques; is it really correct? We claim that the answer is negative. The problem is that existing concepts of correctness do not capture an important sense in which a protocol may be incorrect. Specifically, although protocol goals (e.g., delivering messages) may be attained eventually, the time periods to achieve these goals may increase indefinitely. A notion of correctness which allows one to consider both the probability of reaching a goal as well as the time or computational effort required to achieve the goal is required. We present a novel approach to protocol correctness which unifies functional and performance considerations using a recently proposed probabilistic semantics for programs.

1. INTRODUCTION

The objective of this paper is twofold. First we wish to demonstrate that existing techniques for protocol verification are not sufficient to capture some important aspects of protocol correctness. Given this problem, we will then suggest a unified approach to the verification and performance analysis of protocols, based on a recently developed probabilistic semantics for programs; we feel such a unified approach will be both necessary and sufficient to resolve the problem. The remainder of this section motivates this work by discussing the alternating bit protocol and its correctness. Section 2 then presents an overview of a probabilistic semantics for programs and section 3 outlines how probabilistic semantics might be applied in protocol analysis in order to bridge the traditional gap between the functional and performance analysis of protocols.

1.1. ARE CURRENT NOTIONS OF PROTOCOL CORRECTNESS ADEQUATE

Consider the alternating-bit protocol [7, 1, 2] described in terms of the finite state automaton of figure 1. The left automaton in figure 1 describes the sender protocol while the right automaton depicts the receiver protocol.

---

*This work was supported in part by National Science Foundation Grant NSF MCS-8110319, the Defense Advanced Research Projects Agency and the IBM Corp.

Although the alternating bit protocol in figure 1 delivers messages in one direction only, our remarks concerning this protocol will also apply to the full duplex version of the protocol.

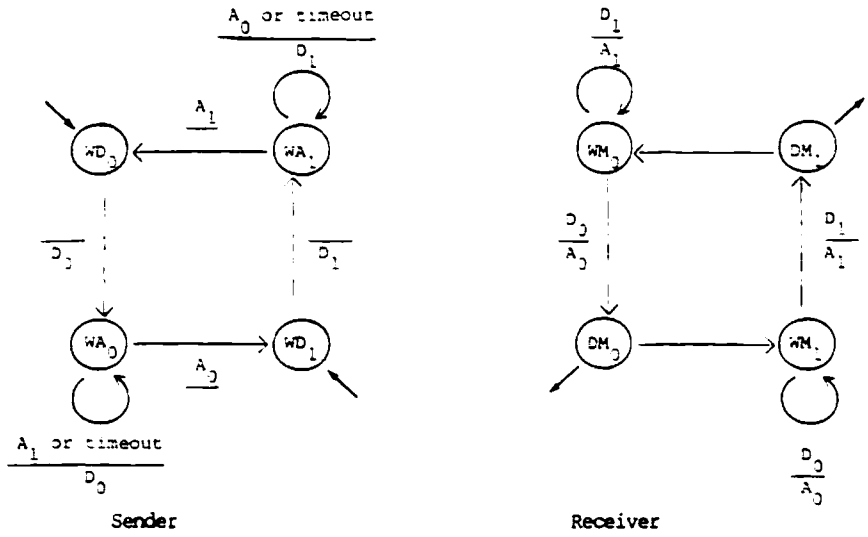

Figure 1-1: THE ALTERNATING-BIT PROTOCOL

The alternating-bit protocol has become the benchmark of protocol verification techniques; it has been verified to be correct using almost every known verification technique. But is it really correct? The answer to this question depends upon one's definition of correctness. Correctness is usually divided into proving safety properties and proving liveliness properties [11, 8, 4, 10, 6]. Safety properties are assertions in the Floyd-Hoare style stating that if anything ever happens it is not going to be "bad". Liveliness properties are assertions that something "good" will actually be achieved. The alternating-bit protocol has been verified to satisfy all reasonable safety and liveliness properties. We do not wish to cast a doubt that the alternating-bit protocol has all the safety and liveliness properties that it should be expected to have. Quite contrary, let us assume that it is both a safe and a live protocol. Does this guarantee that it is "really" correct? That is, is the notion of protocol correctness adequately captured by the notions of safety and liveliness?

1.2. A COUNTER EXAMPLE

To answer the question raised in the previous section, let us assume that

- The medium (including the buffer for incoming messages) is perfectly reliable and does not lose messages.
- When the sender is in the WA_0 or WA_1 state, there is a non-zero probability that the sender times out (thus sending a copy of a data

message) before an acknowledgment arrives from the medium.

With these assumptions let us consider the sequences of messages and acknowledgments in the medium. Both sequences consist of runs of 0's and 1's. Let S and R denote the sender and receiver automata respectively. Let S—>R (respectively, R—>S) denote the sequence of messages (acknowledgments) from S to R (R to S).

Consider now a run of 0's in the message sequence S—>R. The first message marked 0 is received at R which is in the state WM_0. R performs the transition to DM_0 sending an acknowledgment marked 0 back to S, delivers the data message and then moves to WM_1. Subsequent additional copies of the message marked 0 cause R to respond with an acknowledgment. So the run of acknowledgments is equal in length to the run of messages that are acknowledged. Now, when the first acknowledgment is received by S it moves from the state WA_0 to WD_1, where it picks up and sends the first copy of message 1, thus beginning a run of 1's in the S—>R sequence. S then moves to WA_1 where it responds to any acknowledgment of message 0 with a copy of message 1. Therefore, the length of the run of 1's in the S—>R message sequence is at least as long as the previous run of 0's.

Now, each time the timer times out when S is in state WA_1 or WA_0, the length of the respective run in the S—>R sequence increases by 1. This implies that the runs in the S—>R sequence (and therefore in the R—>S sequence) will always eventually increase in size. Therefore, while any safety assertion about the protocol is not violated and any liveliness assertion will eventually become true, the time periods during which the protocol progresses towards achieving the liveliness assertions increase indefinitely.

To summarize, while safety assertions can show that nothing bad will happen and liveliness assertions can show that something good will be attained (eventually), the time periods for the "good" thing to happen may grow, with probability 1, to infinity.

1.3. HOW LONG BEFORE THE EVENTUAL HAPPENS AND WITH WHAT PROBABILITY

Two interesting aspects of the behavior of the alternating bit protocol are illustrated by the above example. First, the "bug" occurs in its worst form when the medium is perfect. Second, the only mechanism to save the day is to guarantee that some fraction of the messages be lost. That is, using a simple queueing model one can prove that a necessary and sufficient condition that the instability discussed above does not exist is that messages should be lost at a rate exceeding the rate at which new copies are generated by the timer. This criteria can be met by selecting an appropriate time-out interval and introducing a probabilistic loss mechanism (e.g., before the sender machine sends a copy of a data message in response to a timeout or the receipt of a duplicate acknowledgment, a coin should be tossed to determine whether or not a copy of the message should be sent.)

In order to have a correctness theory that can adequately capture phenomena as in the example above, it is necessary to incorporate assertions that reflect the probabilistic behavior of the protocol, such as the probabilistic amount of time (measured in the length of some sequence of states or events) needed to reach some goal or the probability that certain phenomena will occur. These requirements are the basis of this paper.

It should, however, be emphasized that the counter example provided above

merely serves to underline some of the issues and problems which cannot be handled by pure functional analysis of protocols. The following claim, we feel, is much more significant (yet less formal). Protocols are, among other things, real-time software. As such, their design is guided by a mixture of performance and functional objectives. Therefore the classical software paradigm: "correctness first then performance" is no longer valid; the new theory must capture this mixture of design considerations in a unified manner rather than draw artificial boundaries.

2. THE PROBABILISTIC SEMANTICS OF PROGRAMS

From the preceding discussion, it is evident that a theory is needed that can adequately capture the probabilistic behavior of a protocol resulting from the non-deterministic nature of the medium and possibly the protocol itself. Recently, Kozen [5] and Ramshaw [9] have developed a semantics of probabilistic programs which we believe can provide the essential link between the functional and performance analysis of protocols. In this section, we present the salient features of probabilistic semantics, taking an intuitive rather than a strictly formal approach; formal descriptions of the semantics can be found in [5] and [9]. It should be observed that while the probabilistic semantics presented here pertain specifically to ALGOL-like programs, the fundamental ideas behind probabilistic semantics extend easily to other models of computation, including finite state machines.

An important idea underlying probabilistic semantics is the consideration of the distribution or measure associated with a program variable rather than the specific values of the variable resulting from a specific execution. The vector $\underline{x}$ containing the program variables is considered to have some joint distribution, μ on input. We can think of μ as a fluid mass distributed over all possible values of $\underline{x}$; the amount of mass in any region is determined by how likely $\underline{x}$ is to have an initial value in that region. A program's execution serves to redistribute this fluid mass over all values of $\underline{x}$. A program can be interpreted as a linear operator which transforms an input distribution μ to a final distribution μ'.

The transformation of μ to μ' (i.e. the redistribution of the fluid mass) is accomplished by the composition of transformations resulting from the execution of the individual program statements. Each program statement type is interpreted as an operator or transformation whose effect is to transform the distribution immediately preceding the statement type. Execution of an assignment statement simply redistributes the fluid mass. For example, if x is the only program variable and is normally distributed around 0 before the statement x:=x+1 is executed, then x is normally distributed around 1 after the assignment statement. An if statement conditioned on the value of $\underline{x}$ splits the mass of $\underline{x}$ in two and each branch of the conditional is executed on one piece of the split mass. In a while loop, the part of the mass not satisfying the while test splits off and the body of the while loop is executed on the remaining mass. The part of a mass emerging from a pass through a while loop that does not satisfy the while test splits off and the process is repeated on the remaining mass. Should any mass get infinitely stuck in the loop, this indicates that for certain input values, the program will never halt.

The formal semantics of Kozen and Ramshaw, which formalize the above intuitive

ideas, are stated in measure theoretic terms.* Let $\underline{x}$ range over a set of values X, μ be an initial measure induced by $\underline{x}$ and S be a program. If S is the program:

$$\underline{x} := F(\underline{x})$$

then S is interpreted as a linear transformation that takes the input measure μ to the final measure $\mu \cdot F^{-1}$, where $\cdot$ is the composition operator. This interpretation agrees with our intuition and states that the probability that $\underline{x}$ has a value in the set A after the execution of S is the same as the probability that $\underline{x}$ is initially (on input) in the set $F^{-1}(A)$, which maps to A under F.

The interpretation of a larger program is constructed recursively. If S is the program $T_1;T_2$ and T_1 and T_2 have interpretations t_1 and t_2 respectively, the the interpretation of S is given by $t_2 \cdot t_1$. Suppose S is the program:

$$\text{if b then } T_1 \text{ else } T_2$$

Intuitively, we know that if the T_1 branch is taken, the initial distribution μ must be conditioned using the information that on entry to T_1, $\underline{x}$ is known to satisfy b. Let B be the set of all values of $\underline{x}$ satisfying b and define the measure e_B on any set A to be:

$$e_B(A) = \mu(A \cap B)$$

It can be shown [5] that the interpretation of S with an initial measure of μ is the transformation:

$$t_1 \cdot e_B + t_2 \cdot e_{-B}$$

Informally, this states that if μ' is the measure resulting from S with initial measure μ, then the result of applying μ' to some set A is equivalent to assuming an initial measure e_B for T_1 and e_{-B} for T_2 and summing the results of applying the resulting final measures of T_1 and T_2 to A.

Finally, the interpretation of the while statement can also be constructed recursively. If S is the program

$$\text{while b do } T_1$$

then the interpretation of S should be the same as for

$$\text{if b then } T_1;\ \text{while b do } T_1$$

which results from "unwinding" one pass through the while loop. This means that the transformation associated with S must be a solution of the transform equation:

$$S = e_{-B} + S \cdot t_1 \cdot e_B$$

A solution to this transform equation can be found using well known methods from functional analysis; these methods are also discussed in [5].

---

*In a probabilistic context, a measure μ associated with the random variables $\underline{x}$ has the following interpretation: if the possible values of $\underline{x}$ range over a set X and B is a subset of X, then $\mu(B)$ is the probability that the random variables $\underline{x}$ have a value contained in B.

3. PROBABILISTIC SEMANTICS OF PROTOCOLS

In order to extend the probabalistic semantics of sequential programs to protocols, several aspects of protocol operation must be considered. First, protocols involve interaction among concurrent computations leading to randomized interleavings of activities. Second, protocols involve an unreliable medium which may (and should) be modeled as a probability transformer. Third, protocols involve timers which may time-out at random times (relative to other on-going activities). It is desirable that the probabilistic semantics of protocols should provide the mechanism to model and analyze these features. In what follows we present a rudimentary description of how this may be achieved. A more detailed description and applications of these ideas will be published in the near future.

To describe an appropriate solution, let us return to the alternating-bit protocol. The protocol consists of three components: the sender, the receiver and the medium. The sender and receiver states and transitions are completely specified by the finite state automaton description. It remains to model the state of the medium. The state of the medium may be described in terms of two ghost buffers one containing the S—>R message stream (represented as a sequence of 0's and 1's) and one containing the R—>S acknowledgment stream.

Let S denote the state of the sender (S assumes the values WD_0, WA_0, WD_1, WA_1) and let R denote the state of the receiver (R assumes the values WM_0, WM_1, DM_0 and DM_1). The state of the medium may be described in terms of the 0-1 sequence-valued ghost variables M and A where M[i] (A[i]) contains the i-th message (acknowledgments) in the medium to be delivered from S to R (R to S). The state of the system is described by the tuple <S,R,M,A>.

The probabilistic approach to semantics considers probability distributions over the states <S,R,M,A> and describes the behavior of the system in terms of a linear probability transformer. Suppose that the system state is distributed according to the probability measure μ. How does the system transform this distribution?

Transitions in the state of the system may be caused by either

1. Sender transitions due to an arrival of new data.
2. Sender transitions due to an arrival of an acknowledgments.
3. Sender transition due to a time out.
4. Receiver transition due to an arrival of a message.
5. Receiver transition due to a delivery of data.

Each one of the above transitions corresponds to a probability transformer. Let T^i i=1..5, be the probability transformer associated with the i-th transformation above and let B_i denote the condition under which transition i is enabled. Let us assume that the delays in the medium and arrivals of messages are exponentially distributed. With these assumptions, it is easy to see that the system transformer T may be expressed as a linear combination

$$T = a_1 T_1 \cdot e_{B_1} + a_2 T_2 \cdot e_{B_2} + a_3 T_3 \cdot e_{B_3} + a_1 T_4 \cdot e_{B_4} + a_5 T_5 \cdot e_{B_5}$$

Where e_B is the conditioning operator of the previous section and a_i is a coefficient representing the probability that the respective transition will fire, once enabled.

It is interesting to note the analogy between functional and performance analysis of protocols. First, the probability transformer of the system is analogous to the transition matrix of the respective Markov process; the existence of a fixed point of the system probability transformer is similar to the ergodicity of the Markov process. Secondly, modular decomposition of correctness proofs [11, 3], i.e. considering the receiver and the sender separately and relating their activities through their interface with the medium, is analogous to the conditioning process which decomposed the system transformer to a sum of respective local operators.

4. CONCLUSIONS

In this paper, we have demonstrated that current protocol verification methods cannot capture certain correctness properties of protocols. We have suggested an approach to protocol verification which captures these properties through a unified consideration of the functional and performance aspects of protocols. Our work in this area is still in its early stages; results from further research concerning the development and applicability of this early work will be reported in the future.

REFERENCES

[1] Bartlett K.A., Scantlebury R.A., Wilkinson, P.T.
A Note on Reliable Full-Duplex Transmission over Half-Duplex Lines.
Communications of the ACM 12, No 5, May, 1969.

[2] Bochman, G.V.
Finite State Description of Communication Protocols.
Computer Networks 2, October, 1978.

[3] Bochman, G.V. and Sunshine, C.A.
Formal Methods in CCommunication Protocol Design.
IEEE Transactions on Communications COM-28:624-631, 1980.

[4] Hailpern, B. and Owicki, S.
Verifying Network Protocols Using Temporal Logic.
In Trends and Applications Symposium. NBS, 1980.

[5] Kozen, D.
Semantics of Probabilistic Programs.
In Proceedings of the 20-th Symposium on the Foundations of Computer Science. IEEE, October, 1979.

[6] Lamport, L.
Somethime is Sometimes Not Never: A Tutorial on The Temporal Logic of Programs.
In Proceedings of the Seventh Annual Symposium on Principles of Programming Languages. ACM, 1980.

[7] Lynch, W.C.
Reliable Full-Duplex Transmission over Half-Duplex Telephone Lines.
Communications of the ACM 11, No 6, June, 1968.

[8] Manna, Z. and Pnueli, A.
Verification of Concurrent Programs: The Temporal Framework.
Technical Report, Computer Science Department, Stanford University, 1981.
Report No. STAN-CS-81-836.

[9] Ramshaw, L.H. .
Formalizing the Analysis of Algorithms.
PhD thesis, Computer Science Dept., Stanford University, 1979.

[10] Schwartz, R. and Mellier-Smith, M.
Notes on a Temporal Logic Specification of the Alternating Bit Protocol.
Technical Report, Computer Science Lab, SRI Int., 1980.

[11] Sunshine, C.A. (ed.).
Communication Protocol Modelling.
Artech House, 1981.