

A UNIFIED METHOD FOR THE SPECIFICATION AND VERIFICATION OF PROTOCOLS*

GREGOR V. BOCHMANN and JAN GECSEI
Département d'informatique, Université de Montréal
Montréal, Canada

Verification of communication protocols usually involves two parts: a state-machine analysis of the control structure and proving some assertions about the semantic content of the protocol's actions. The two parts are traditionally treated separately. This paper suggests that the two approaches are not independent but rather complementary. It introduces a unified model for protocols (and generally cooperating distant subsystems) encompassing both aspects. The method is demonstrated on three different descriptions of the same protocol, each with a different tradeoff between state machine and programming aspects. Verification of partial and full correctness is carried out in terms of the three descriptions.

1. INTRODUCTION

Experience with design and logical verification of communication protocols indicates that various techniques are suitable for the verification of different properties of the same protocol. All known verification techniques derive in some way from two fundamental approaches: the state machine approach [1,2] and the programming language approach. [3,4] The first of these has been used when the properties of the protocol to be verified are such as the absence of deadlocks or undesired loops or proper sequencing of operations. The programming language approach is used with properties involving counting and, in general, in cases when the state machine representations would become too complex (involve too many states).

The state-machine techniques use always some form of reachability analysis, whereas the programming language method relies on proving assertions and invariants [5] and normally does not address the question of reachability or termination.

It would seem, at first, that there is little connection between the state-machine and programming language approaches to verification. This is partly because both methodologies have their own established formalism, quite different one from another. Thus, attempts to establish a bridge between the methodologies may be frustrated by the necessity to pass from one formalism to the other, which is not always trivial.

It is our belief that the two approaches to verification are not independent, but rather complementary techniques. In order to benefit maximally from both methods, they should be used together; but it is first necessary to create a model that incorporates both the state machine and programming language formalisms. Such a model is described in sections 2 and 3. We believe that this model is widely applicable to the specification and verification of systems of communicating processes. In order to show its usefulness, we have chosen a particular system, a simple data communication protocol working over an unreliable transmission medium, for which we present three different specifications in section 4. In section 5 we demonstrate how some correctness proofs can be carried out for the three descriptions.

*This work has been partly supported by the National Research Council of Canada.

2. THE BASIC MODEL

In a recent paper, Keller [6] has proposed a model for the representation of parallel programs. His model is essentially a Petri net [7] composed of a set of places and transitions complemented with a set of variables X . Each transition t in the net has associated with it an enabling predicate P_t , depending on some variables of X , and an action A_t , assigning new values to some variables of X . The state of the modeled system is determined by the number of tokens that reside in different places and the values of the variables. A certain transition t of the system is enabled when all its input places have at least one token (standard rule for Petri nets) and its enabling predicate P_t is true. When a transition is enabled, it may fire, i.e. the corresponding action A_t is executed, and the tokens are redistributed according to the rules of Petri nets.

In the original model all transitions and actions are assumed to be instantaneous, which implies their mutual exclusion.

Keller's model is intuitively appealing since it is capable of naturally representing some important aspects of the systems being modeled:

- control structure is represented by the inter-connection of places, transitions and some variables of the set X
- semantic structure is represented by the variables, predicates and actions associated with transitions
- parallelism and coordination can be modeled by having several transitions enabled at the same time. The number of tokens in the model is generally not limited.

3. THE EXTENDED MODEL

In Keller's model each variable can, in principle, be affected by all transitions in the system. For the description of distributed systems which consist of several communicating subsystems located at different points in space, it seems to be natural that local variables of a given subsystem can only be affected by the transitions of that subsystem. We therefore extend Keller's model to include the possibility of having several disjoint subsystems and some means of communication between them as follows.

A system S (i.e. parallel program) is composed of a number of subsystems S_1, S_2, \dots, S_n . Each subsystem, separately, is modeled by the formalism of the previous section. If the set of variables of subsystem S_i is called X_i (the local variables of S_i), then the predicates and actions (called local actions) of the subsystem X_i only refer to these local variables.

For the interaction of different subsystems, each subsystem may contain certain distantly initiated actions. Like the local actions, they may assign new values to the local variables; however, they are not associated with a given transition of the subsystem. Distantly initiated actions are executed some finite time after they have been initiated by a distant subsystem; this is done by the execution of an initiating statement in a local action of the distant subsystem. The initiating subsystem may pass value parameters for the execution of the distantly initiated action. All actions in a subsystem are executed in mutual exclusion.

This form of interaction between subsystems seems to capture the essential properties of subsystem communication through the exchange of messages. In fact, the initiation of an action in a distant subsystem corresponds to the sending of a message (the action parameters are the message content), and the execution of the distantly initiated action corresponds to the receiving of the message by the distant subsystem.

We note that the state of the system, at a given instant in time when no action is being executed, is given by the states of all subsystems, i.e. their token distribution and variable values, and the set of distant action initiations which have not yet been executed. The latter set can be understood as the state of the "communication medium", or the messages "in transit".

We also remark that the set of variables X_i together with all actions defined in S_i constitute an abstract data type with mutual exclusion of the actions. [8]

For the specification of the variable declarations, predicates and actions of a subsystem, we use a notation close to the programming language Pascal. [9] Initiation of a distant action can be achieved by the primitive INITIATE < name, p_1, \dots, p_k > appearing as a statement in a local action, which specifies the name of a unique distantly initiated action and k parameter values. We note that the initiating action does not wait for the completion of the initiated action, and that the order of execution of several distantly initiated actions may be different from the order in which they were initiated.

4. EXAMPLES


In this section we show the flexibility of the extended model by giving three descriptions of the same protocol: the first and second minimizing the number of places and variables respectively, and the third having a certain balance between them.

The protocol we use is essentially the "alternating bit" protocol of Bartlett [10] which can be summarized as follows:

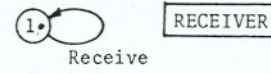
- It is a point-to-point protocol using the communication medium alternatively in both directions.
- In contrast to [10] we suppose data transfer in one direction only, from the SENDER subsystem to the RECEIVER subsystem.

- The SENDER waits for an acknowledge message before the next data message is sent.
- The protocol recovers from transmission errors detected by a redundancy check, and from lost messages through a time-out mechanism in the SENDER. In both cases, retransmission of the data message occurs.

4.1 One-place description

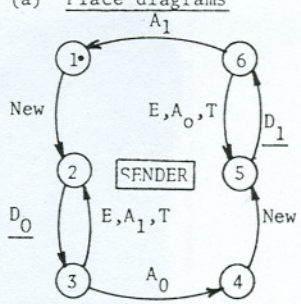
- (a) Place diagram  SENDER
- Initial state — seq=1; ack=1
- (b) Variables: Same as three-place description
- (c) Actions

Transition	enabling predicate	action
Send	ack≠none v tout=true	if ack=seq then begin new(data); seq:= seq+1(mod2); end; INITIATE (transD,seq,data); ack:=none; time:=t ₀ ; tout:=false;
Clock transA(p:(0,1)) } same as three-place description		

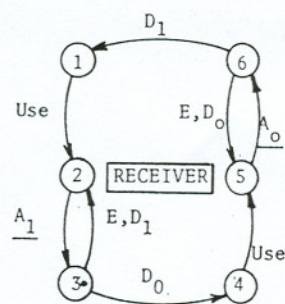
- (a) Place diagram  RECEIVER
- Initial state — exp=1; seqnb=none
- (b) Variables: Same as three-place description
- (c) Actions

Transition	enabling predicate	action
Receive	seqnb≠none	if seqnb=exp+1(mod2) then begin use(data); exp:=exp+1 (mod2); end; INITIATE (transA, exp); seqnb:=none;
transD(p ₁ :(0,1),p ₂ :...) same as three-place description		

4.2 Six-place description

- (a) Place diagrams
- 

Initial state: — token in 3
clock — tokens in 1,7

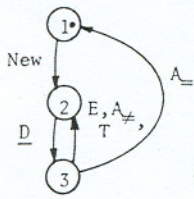


Initial state: — token in 3
seqnb=none
- (b) Variables: same as in three-state description except that seq and exp are no longer needed as a consequence of the "unfolded" place diagrams.
- (c) Actions: There would be an action (possibly empty) associated with each transition. We do not include a detailed list, since they are analogous to those of the 3-place description.

4.3 Three-place description

(a) Place diagram

SENDER



Initial state:
 - tokens in 1,4
 - seq = 1

Clock

(c) Actions

transition	enabling predicate	action	meaning
New	true	new(data);seq:=seq+1(mod2);	get new data from user
<u>D</u>	true	INITIATE(transD,seq,data); ack:=none;time:=t ₀ ; tout:=false;	transmit message (seq,data)
A ₌	ack=seq	;	reception of expected acknowledge
A _≠	ack=seq+1(mod2)	;	reception of wrong acknowledge
E	ack=error	;	error in received acknowledge
T	tout=true	;	timeout has occurred
Clock	true	time:=time-1;if time=0 then tout:=true;	timer action
distantly initiated action transA (p:(0,1))			depending on the transmission medium, one of the following will occur:
case transmission of correct:ack:=p;			acknowledge received
erroneous:ack:=error;			erroneous reception
loss ;;			message lost

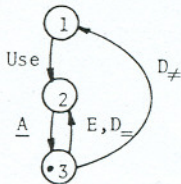
(b) Variables

Meaning

seq: (0,1) sequence number of message sent in this cycle
 ack: (0,1,error,none) acknowledge from receiver
 data: . . . data to be transmitted
 tout: boolean time-out has occurred
 time: integer timer count

(a) Place diagram

RECEIVER



Initial state
 - token in 3
 - exp = 1
 - seqnb = none

(c) Actions

transition	enabling predicate	action	meaning
Use	true	use(data);exp:=exp+1(mod2);	give data to user
<u>A</u>	true	INITIATE (transA, exp); seqnb:=none;	transmit message (exp) (= acknowledge)
D _≠	seqnb=exp+1(mod2)	;	reception of message with expected sequence number
D ₌	seqnb=exp	;	reception of message with wrong sequence number
E	seqnb=error	;	error in received message
distantly initiated action transD(p ₁ :(0,1);p ₂ :...)			depending on the transmission medium, one of the following will occur:
case transmission of correct:seqnb:=p ₁ ; data:=p ₂ ;			message received
erroneous: seqnb:=error			erroneous reception
loss ;;			message lost

(b) Variables

Meaning

exp: (0,1) opposite of expected sequence number of message received in this cycle
 seqnb: (0,1,error,none) sequence number of received message
 data: . . . data in received message

4.4 Comments

The purpose of the preceding examples is to demonstrate that places and variables are complementary means of representing the state of communicating subsystems. The correctness proofs outlined in the following section are based on both aspects of the formalism we use.

We note, however that in these examples the full power of Petri nets is not used; it is not clear to us at this point whether this power is useful in modeling communication protocols. The idea of using finite state machines and variables for protocol description is not new; [11] however, our approach incorporates also a means for describing communications, which leads to a unified proof methodology.

It should be clear also that the concept of distantly initiated actions can serve equally for modeling of more general communication systems such as the datagram service, or communicating processes in operating systems.

5. VERIFICATION

We demonstrate in this section how the modeling technique described previously can be used for the verification of different properties of a protocol such as absence of deadlocks, liveness, cyclic behavior, partial and full correctness of the global system. Of course these properties are not mutually independent; however, the first four, generally, are necessary conditions for the last one.

Deadlock-freeness, liveness and cyclic behavior are best derived from an analysis of possible transitions of the global system i.e. the reachability analysis. [1,2] This in turn requires taking into account the control structures of each subsystem, certain constraints on the order in which transitions and distantly initiated actions can be executed, and some assertions on program variables.

Verification of partial correctness [5] will correspond in this paper to finding out whether and in which circumstances the sender subsystem (and its user) can "know" that all data obtained from the user have been delivered correctly and in sequence to the user in the receiver subsystem. This knowledge can be expressed by the predicate

$$P_1 : \text{Producer-sequence} = \text{Consumer-sequence.}$$

We say that the sender is in a complete state when this state implies P_1 . Partial correctness of the system means then the existence of a complete sender state, and full correctness means that such a state is always reached after a finite amount of time (liveness of the complete sender state and absence of deadlocks).

We show in section 5.2 that for the three-state description (see section 4.3) of the "alternating bit" protocol the sender state "token in place 1" is complete. Similarly for the one- and six-place descriptions the sender states "ack = seq" and "token in place 1 or place 4" respectively, are complete.

5.1 Possible transitions of the global system in the three-place model

Before constructing a transition graph, we have to point out the existence of the following constraint: the predicates and actions of the sender subsystem are defined such that after the execution of transition \underline{D} (containing $ack := none$), the transitions \underline{A} , \underline{A}_\neq or \underline{E} can only become enabled after execution of the distantly initiated action $transA$ with correct or erroneous transmission. A similar

constraint holds for the receiver. We also see that the time-out transition can only occur after the timer has been set by transition \underline{D} and t_0 clock transitions have occurred.

We can now determine the possible transitions of the global system as shown in the diagram of fig. 1. Each state $\langle p_1, p_2 \rangle$ action of the global system is characterized by the active places p_1 and p_2 (containing a token) of the sender and receiver subsystem respectively and, possibly, by a distantly initiated action not yet executed. The details of deriving such diagrams have been presented elsewhere. [2] Briefly, it is based on the control structure of the subsystems, on the constraints mentioned above, on the fact that the actions $transA$ and $transD$ are initiated (only) by the \underline{A} and \underline{D} transitions of the receiver and sender respectively, and on the initial state of the system.

We have assumed that the time-out delay t_0 could be chosen such that the time-out transition T will only occur after a transmission loss has occurred. This clearly depends on the execution speeds and delays of the different transitions and distantly activated actions. We have not included these considerations [1] in our model.

We can conclude from fig. 1 that the constraints mentioned above do not introduce any deadlock (each state has a successor) and that the system shows a cyclic behavior such as expected for a data transmission protocol.

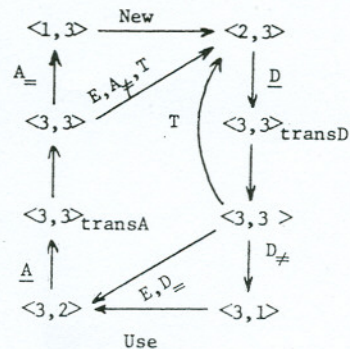


Fig. 1. Possible transitions of the global system (three-place description)

5.2 Verification of partial and full correctness of the three-place description

We can establish the following assertions

$$AS_1 : \text{sender token in place 3} \wedge \text{sender.ack}=0 \text{ or } 1$$

$$\Rightarrow \text{sender.ack} = \text{receiver.exp}$$

$$AS_2 : \text{receiver token in place 3} \wedge \text{receiver.seqnb}=0 \text{ or } 1$$

$$\Rightarrow \text{receiver.seqnb} = \text{sender.seq} \wedge \text{receiver.data} = \text{sender.data}$$

which are used below for proving the partial and full correctness. Assertion AS_1 follows from the fact that when $ack = 0$ or 1 in place 3 then the action $transA$ must have been executed since the sender has entered place 3. However, the receiver uses the value of exp as an effective parameter for

initializing the action *transA* and the receiver could not have done any further transition (see fig. 1) thus leaving the variable *exp* unchanged. The assertion AS_2 can be shown similarly.

Now we define the following global predicates:

P_1 : Producer-sequence=Consumer-sequence (as above)

P_2 : Producer-sequence=Consumer-sequence
| sender.data (where the " | " means concatenation)

P_3 : sender.seq = receiver.exp

and establish the invariant assertion

$I : (P_1 \wedge P_3) \vee (P_2 \wedge \neg P_3)$, which is proved by induction over the number of transitions executed. Initially $(P_1 \wedge P_3)$ holds, which implies I . Suppose now that I holds in some given state of the system; we have to show that I also holds after one of the subsystems has executed a transition or a distantly activated action. We note that the distantly activated actions do not affect the predicates P_1 , P_2 or P_3 , neither do the transitions, except the *New* transition of the sender and the *Use* transition of the receiver.

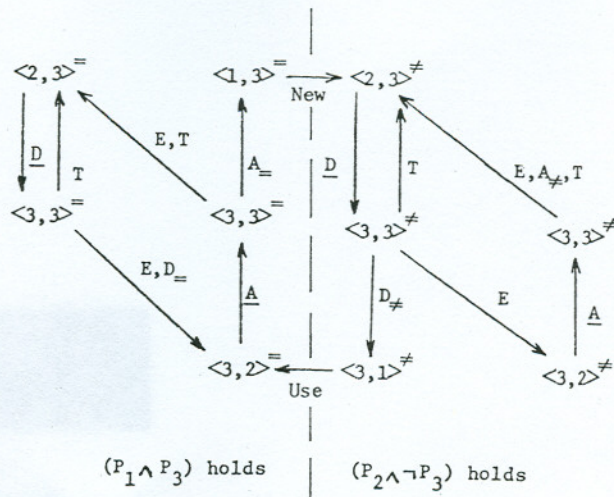


Fig. 2. Possible transitions of the global system (three-place description, distinguishing states with respect to P_3)

The following arguments show that I is invariant in respect to the execution of the transition *New*; similar arguments apply for the transition *Use*. From AS_1 and the enabling predicate of the transition A_+ in the sender follows that P_3 holds when a token is in place 1. Together with I , this implies that P_1 holds in place 1. We now consider the axiomatic definition [5]

Producer-sequence
Q
Producer-sequence | data {new (data)} Q

for the procedure *new*, which means that for proving an assertion Q to hold after the execution of the statement "new (data)", it is sufficient to prove that Q' holds before the execution of the statement, where Q' is obtained from Q by substituting "Producer-sequence | data" for each occurrence of "Producer-sequence" in Q . This definition, together with the form of the action associated with the transition *New* shows that $(P_2 \wedge \neg P_3)$ holds after the execution of *New*, which implies that I holds, too. Therefore I is invariant in respect to the transition *New*.

As mentioned above, the invariant I implies that P_1 holds when the sender is in place 1. Therefore this is a complete state, which implies the partial correctness of the protocol.

Now, in order to demonstrate its full correctness, we have to show that the complete sender state indicated above is live. This can be seen from fig. 2 which shows the diagram of possible transitions of the global system, where, in contrast to fig. 1, we distinguish the states for which P_3 holds (indicated by a "=") and those for which $\neg P_3$ holds (indicated by a "≠"). The transition diagram shows that the state $\langle 1,3 \rangle$, corresponding to the complete state of the sender, lies on the main loop which is always followed when the transmission medium works correctly. We note that in this case the transitions A_+ and D_+ will never be blocked (see AS_1 and AS_2). Therefore the complete sender state is live as long as there is no permanent malfunction of the transmission medium.

5.3 Verification of the one-place protocol description

The verification follows the same lines as for the three-place description. The assertions corresponding to AS_1 and AS_2 are

AS_1' : sender.ack = 0 or 1

\Rightarrow sender.ack = receiver.exp

AS_2' : receiver.seqnb = 0 or 1

\Rightarrow receiver.seqnb = sender.seq \wedge
receiver.data = sender.data

and the invariant I is the same as before.

We note that the diagram of possible transitions for the global system does not contain much information in this case, since each subsystem has essentially only one place. This implies, in particular, that the proof of the liveness of the complete sender state is not as clear as in the case of the three-place description.

5.4 Verification of the six-place protocol description

The verification follows similar lines as for the one- and three-place descriptions. The analysis of possible transitions of the global system yields the diagram of fig. 3. The only assertion used is AS_2 : "receiver token in place 1 or 4 \Rightarrow receiver.data=sender.data" and corresponds to assertion AS_2 of the three-place description. There is no invariant, but either P_1 or P_2 hold depending on the places of the sender and receiver tokens (see fig. 3). From this follows that the sender is in a complete state when a token is in place 1 or 4.

We note that the diagram of fig. 3 is equivalent to the one of fig. 2, except that for the six-place description each state in fig. 2 is replicated twice, once for the value of *seq* = 0 and once for *seq* = 1. We see that in this case [12] the reachability analysis that yields fig. 3 provides the proof of the liveness of the complete sender state, as well as the essential part of the "partial correctness" proof.

6. CONCLUSIONS

We have shown that the two complementary approaches of state machine models and the use of variables can be combined into a unified method for the specification and verification of systems of cooperating subsystems. Our unified model includes also the con-

REFERENCES

- [1] P.M. Merlin, A methodology for the design and implementation of communication protocols, IEEE Transactions on Comm., Vol. COM-24, 1976, 614-621.
- [2] G.V. Bochmann, Finite state description of communication protocols, Publication # 236, Dép. d'Informatique, Univ. de Montréal, July 1976.
- [3] G.V. Bochmann, Logical verification and implementation of protocols, Proc. Fourth Data Communications Symposium ACM/IEEE, 1975.
- [4] N.V. Stenning, A data transfer protocol, Computer Networks 1 1976, 99-110.
- [5] C.A.R. Hoare, An axiomatic basis for computer programming, CACM, 12, 1969.
- [6] R.M. Keller, Formal verification of Parallel programs, CACM, 7, 1976, 371-384.
- [7] A.W. Holt and F. Commoner, Events and conditions, in Project Mac conference on Concurrent Systems and Parallel Computation, June 1970.
- [8] B.H. Liskov and S.N. Zilles, Specification techniques for data abstractions, IEEE Trans. on Software Engineering, SE-1, p. 7, 1975.
- [9] K. Jensen and N. Wirth, Pascal user manual and report, Springer Verlag, Berlin, 1974.
- [10] K.A. Bartlett, R.A. Scantlebury and P.T. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, CACM 12, 260, 1969.
- [11] A.S. Danthine, J. Bremer, An axiomatic description of the transport protocol of Cyclades, Professional Conference on Computer Networks and Teleprocessing, TH Aachen, March 1976.
- [12] G.V. Bochmann, Communication protocols and error recovery procedures, Proc. ACM Inter-process Communications Workshop, March 1975. Op. Syst. Review, Vol. 9, No. 3, 45-50.

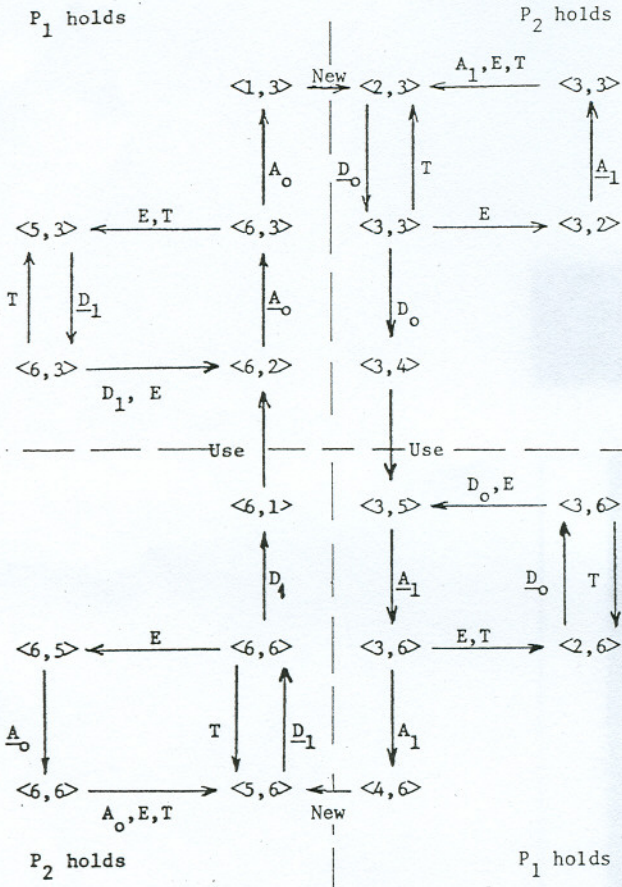


Fig. 3. Possible transitions of the global system (six-place description)

cept of distantly initiated actions, which seems to be useful for modeling the communication of subsystems through the exchange of messages.

We have demonstrated the flexibility of the model by giving three different specifications for the same simple protocol. We believe that the model can also provide a natural description of more complex protocols. For example, the opening and closing of connections are usually described by a state machine model, whereas the data transfer phase is described by a program model with variables. [4] With our model, both aspects could be described in a unique specification.

For the verification, the two aspects of our model complement one another. As shown in the example in the previous sections, the program aspect provides assertions for correctness proofs, whereas the state machine aspect provides useful information for the former and facilitates the proof of liveness or absence of deadlocks.

There is clearly a tradeoff between the complexity of the state machine and program aspects of the specification, as can be seen, for example, from the comparison of the one-place and six-place descriptions. Since reachability analysis of state machines seems to be more amenable to algorithmic methods than verifying (and finding) program assertions, the above tradeoff may have important implications for future automated methods of protocol verification.