Protocol Specification, Testing, and Verification VIII
S. Aggarwal and K. Sabnani (Editors)
Elsevier Science Publishers B.V. (North-Holland)
IFIP, 1988

Verifying a CSMA/CD-protocol with CCS

Joachim Parrow*
Swedish Institute of Computer Science
Box 1263, S-164 28 Kista, Sweden

Abstract

We use the layered structure of communication protocols to derive a verification methodology in CCS. This methodology is applied to the Media Access Sublayer in a Carrier Sense, Multiple Access with Collision Detect protocol. The formal verification step is performed automatically by a program. We discuss the limitations of the methodology and the deficiencies (for verification purposes) of the official protocol standard definition.

1 Introduction

The purpose of this paper is twofold. First, we want to present an introductory example of protocol verification with CCS. Second, we want to examine the synchronisation properties of a CSMA/CD protocol. In this section we describe how communication protocols are usually structured, and how this structure relates to formal verification in CCS. The reader should be familiar with the basics of CCS as presented in [1].

A *communication protocol* is a procedure used by computers to communicate with one another. As a means of reducing the complexity of protocols, the concept of modular, or layered, design has gained wide acceptance. Most communication networks are organised as a series of *layers*. The layers are distributed over all computers in the network. The purpose of each layer is to offer certain *services* to the layers above it. A particular layer accomplishes this by using the services of the layers below it.

As an example, consider ISO:s Open System Interconnection Reference Model (the *OSI-model*). The *physical layer* is concerned with transmitting data bits over a communication medium. The task of the *data link layer* is to take such a transmission facility and transform it into a line that, to the *network layer*, appears free of transmission errors. The network layer controls the routing of messages between different *hosts* (computers). The control of data transportation from source to end destination is performed by the *transport layer*. This layer can establish and destroy connections between processes running on different hosts.

A layer can be viewed in an abstract way, as in figure 1. A layer n consists of *protocol entities* communicating over a medium, which is actually the set of lower layers. The rules and conventions used in the interaction between the protocol entities to provide the service of the layer constitute the *layer n protocol*. In reality, no data are directly transferred between the protocol entities. Instead, data and control information are passed to the $(n-1)^{th}$ layer in the same host. This procedure is repeated for each layer, until the lowest layer (which might be an electrical wire) is reached.

The *service specification* of layer n is the input/output behaviour of the layer as seen from layer $n+1$. It is usually based on a set of *service primitives* which, in an abstract

*This work was partly supported by the Swedish board of Technical Development (STU) under contract 82-3406.

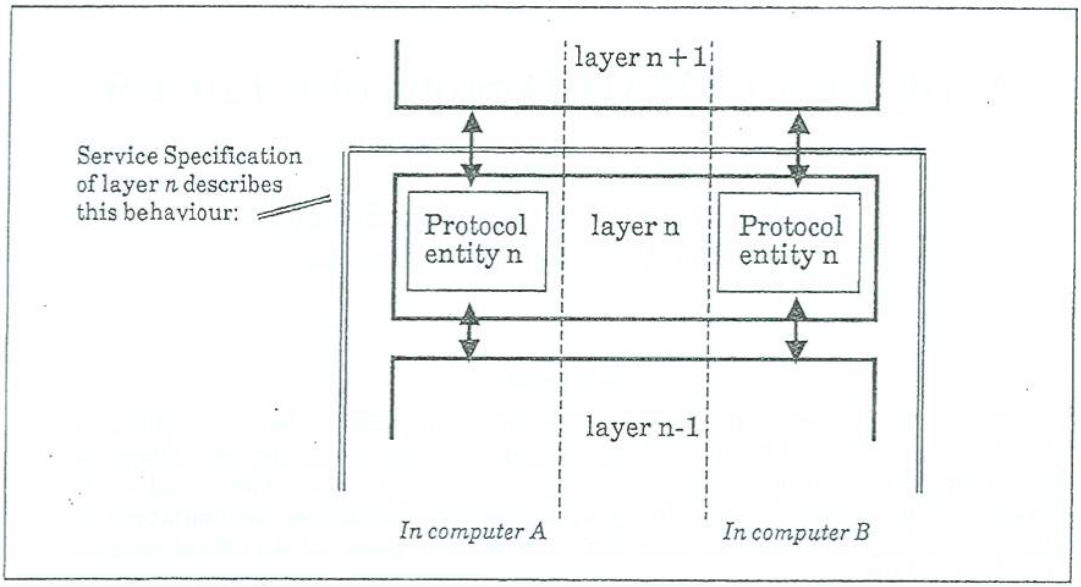

Figure 1: An abstract view of a layer

way, describe the operations at the interface through which the service is provided. Not that a service primitive is always a communication event between two processes in the sam computer.

In CCS, service primitives correspond to observable *actions*. A service specificatio can be defined by an *agent* over such actions. In a data link protocol, for example, th service primitives would be *sendmessage*(m) and *receivemessage*(m) (here m is an objec of a suitable datatype "message"). A simple service specification might be that the laye should behave as a perfect buffer. In CCS, this can be formally defined:

$$ServiceSpec = sendmessage(m).receivemessage(m).ServiceSpec$$

The service specification gives no guidance on how to actually implement layer n. Thi guidance is, to a certain extent, provided by the *protocol specification* of layer n. Th protocol specification lists the protocol entities (in the general case there may be more tha two) and defines the interconnection structure. Furthermore, it defines the input/outpu behaviour of each entity. Usually, it also contains the specification of the service provided b the layers below, i.e. the service specification of layer $n-1$. This abstract "implementation should not be confused with the real implementation of the entities, i.e. program coding o hardware implementation.

In CCS, the input/output behaviour of the protocol entities can be defined by agents the actions of these agents are either to accept service primitives from above, i.e. layer primitives, or to issue service primitives to below, i.e. layer $n-1$ primitives. As an exam ple, consider a simple sender of a data link protocol: the sender accepts a message throug the primitive *sendmessage* (a layer n primitive) and repeatedly transmits it through *trans mitmessage* until an acknowledgment through *acknowledgmessage* (layer $n-1$ primitives arrives. The CCS specification is:

$$\begin{cases} Sender & = & sendmessage(m).Sender'(m) \\ Sender'(m) & = & transmitmessage(m).(Sender'(m) + acknowledgmessage.Sender) \end{cases}$$

A *verification* of the layer n protocol is a formal proof that the layer n protocol spec ification implies the layer n service specification. In CCS, this means to verify that th

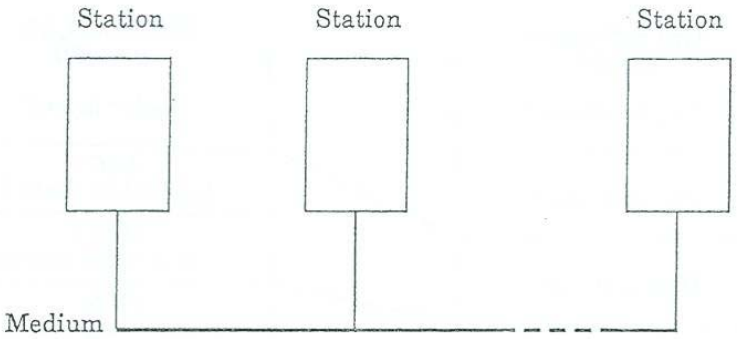

Figure 2: A network with several stations interconnected through a shared medium

agent corresponding to the service specification is observation equivalent with the protocol specification. The protocol specification is obtained by composing all protocol entities in parallel with the service specification of layer $n-1$, and restricting on the layer $n-1$ service primitives (these are neither observable nor accessible from layer $n+1$).

In short, the methodology for verification of protocols in CCS is as follows:

1. Define a set of actions L_n where each action corresponds to a service primitive of layer n. Similarly, define L_{n-1} as the set of actions corresponding to service primitives of layer $n-1$.

2. Define the following CCS agents:

   SS_n The service specification of layer n (this agent has sort L_n).

   SS_{n-1} The service specification of layer $n-1$ (this agent has sort L_{n-1}).

   $PE_1, \ldots, PE_k$ One agent for each of k protocol entities (each agent has sort $L_n \cup L_{n-1}$).

3. Prove in CCS that

$$SS_n \approx (PE_1 \mid \cdots \mid PE_k \mid SS_{n-1}) \setminus L_{n-1}$$

The actual verification is step 3, which can be done completely within the CCS formalism. If all the involved agents are finite state, the observation equivalence is decidable in polynomial time ([2]). We have developed a program to perform this formal verification automatically. From a practical point of view, the first two steps (the description of service primitives and protocol behaviour in CCS) are equally important and often quite difficult. These descriptions should be faithful to existing protocol definitions which are not necessarily expressed in a formally rigid way. Thus it is not always possible to formally verify that steps 1 and 2 are done correctly.

We will devote the next section to an example of protocol verification: the media access sublayer of a CSMA/CD protocol.

2 The CSMA/CD Protocol

The CSMA/CD (Carrier Sense, Multiple Access with Collision Detect) protocol is intended for use in high speed local area networks. Several stations are connected to a shared medium, which can contain at most one message at a time (cf. figure 2). When one station wants to communicate with another, it transmits a message on the medium. If, after initiating a transmission, the message collides with that of another station (i.e. another station

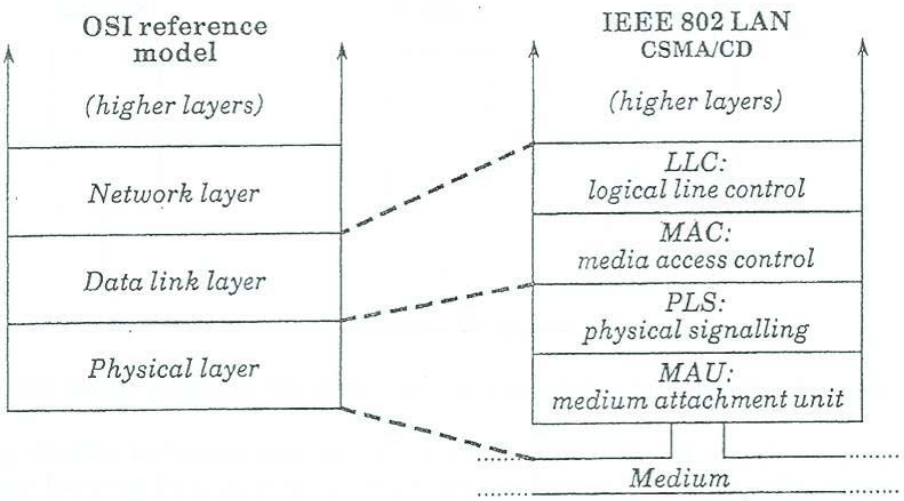

Figure 3: Overview of CSMA/CD

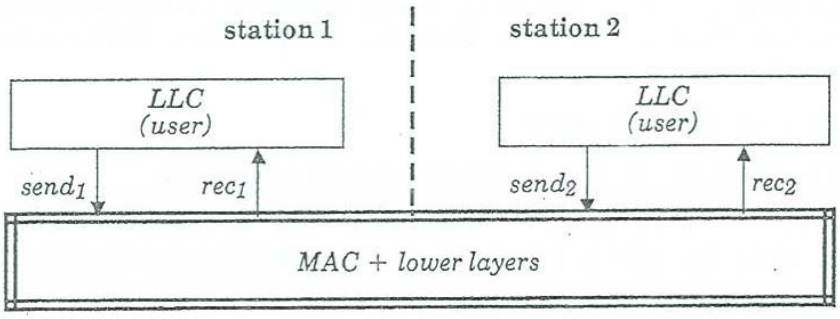

Figure 4: MAC sublayer service overview

attempts to transmit at the same time), then both messages are lost. Following a collision, each station waits for a random amount of time before attempting to transmit again. We base our service and protocol specifications on the official standard definition ([3]), and defer a discussion on the accuracy of our CCS model to section 4.

An overview of the structure of the protocol, and the relation with the OSI model is shown in figure 3. We choose to study the Media Access Control (MAC) sublayer. The MAC communicates with the Logical Line Control (LLC) by accepting or delivering messages. After accepting a message, the MAC repeatedly transmits it to the PLS, until it is delivered intact, i.e. no collision occurs during transmission. The PLS notifies the MAC with a "collision detect" signal in case of collision.

2.1 Service Specification

For simplicity, we will consider the case when two identical stations are connected. We will ignore the actual message contents and focus on the synchronisation properties.

The service provided by MAC is error free bidirectional communication. We model this as follows: MAC accepts messages from LLC on the channel *send* and delivers messages to LLC on the channel *rec*. We use indexing to distinguish between channels in different stations.

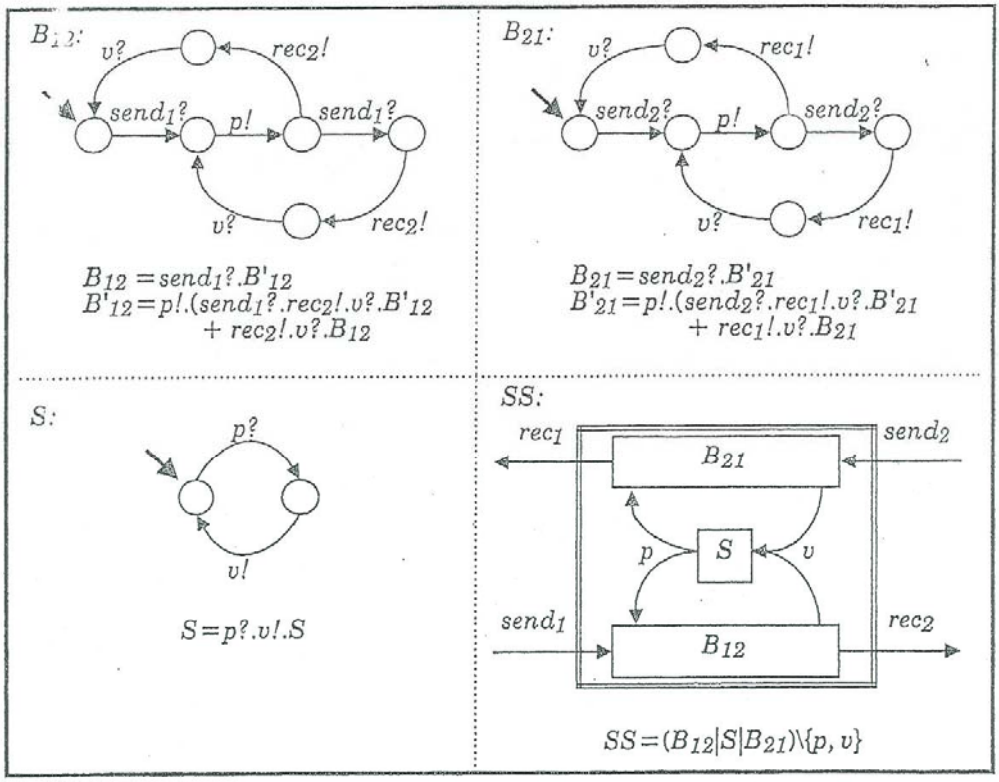

Figure 5: MAC sublayer service specification

Also, we distinguish output events on a channel by appending an exclamation mark (!) to the channel name, and input events by appending a question mark (?). Thus, the service primitives are $send_1?$ and $rec_1!$ in one station, and $send_2?$ and $rec_2!$ in the other. Figure 4 gives an overview of the service. We describe the error free bidirectional communication by one unidirectional buffer in each direction. The official standard definition makes no attempt to further define the synchronisation properties of the service. However, we make the following observations:

- There can be at most two outstanding messages (i.e. messages that have been sent but not received) in every buffer.
- The buffers are not completely independent. They share a common critical resource (namely the medium). Message transport can only occur after a buffer has gained access to this resource. There can be two outstanding messages in a buffer only if that buffer has access to the resource.

We model the service specification by two unidirectional buffers, each with a capacity of two messages. B_{12} transports messages from station 1 to station 2, and B_{21} transports messages from station 2 to station 1. The two buffers are synchronised with a semaphore S. After accepting a message in one station, a buffer must perform a p-operation (waiting on the semaphore) before delivering a message or accepting a second message. A v-operation (signaling on the semaphore) is performed after every message delivery. The service specification SS (cf. figure 5) is expressed as

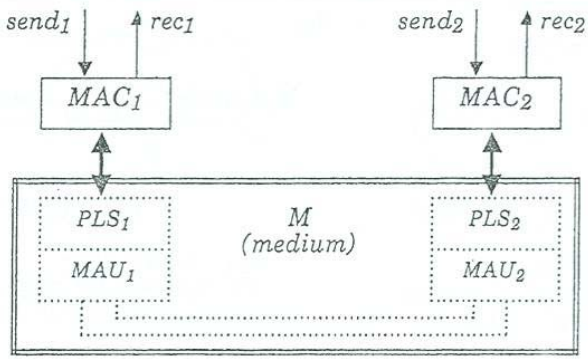

Figure 6: MAC protocol specification overview

$$SS = (B_{12} \mid S \mid B_{21}) \setminus \{p, v\}$$

2.2 Protocol Specification

The protocol specification for the MAC sublayer consists of two MAC entities (one in each station) called MAC_1 and MAC_2, interconnected by a medium M. The medium actually consists of all sublayers below MAC (figure 6). The two MAC stations are identical; as in the service specification we use indexing to distinguish between events in different stations. We must formulate the behaviour of MAC in terms of communication events with LLC and M. The communication events with LLC are the *send?* and *rec!* events used in the service specification. The communication events with the medium should include message transmission, message reception and collision detection. Since a collision may be detected in the middle of a transmission, a transmission can not be a single event. We use the following actions to describe the events:

b The MAC begins message transmission to the medium.

e The MAC terminates message transmission to the medium.

br The medium begins message delivery to the MAC.

er The medium terminates message delivery to the MAC.

c The MAC is notified that a collision has occurred on the medium.

The official standard can not be used directly to derive a CCS specification, since it is not formulated in terms of such events. Our understanding of the standard yields the model as described in figure 7. For clarity we have omitted the indices; in MAC_1 all actions should be indexed by 1, and in MAC_2 indexed by 2.

Initially, a message from LLC (*send?*) may be accepted. The MAC initiates transmission (*b!*), unless a message is in the process of being received. If the transmission is successfully terminated (*e!*), then a new message may be accepted and the process is repeated. If a collision occurs (*c?*) before termination, MAC attempts retransmission after a period of waiting. In all states (except when a message is being transmitted), a message may be received (*br?*). Following such an event, MAC may not begin any transmission until all of the message has been received (*er?*) and the LLC has been notified (*rec!*). However, MAC may accept a transmission request from LLC (*send?*) in any state, unless such a request is pending.

The operation of the medium M in terms of these actions is as follows: initially, M may accept a transmission from one of the MAC:s ($b_1?$ or $b_2?$). Assume $b_1?$; the behaviour

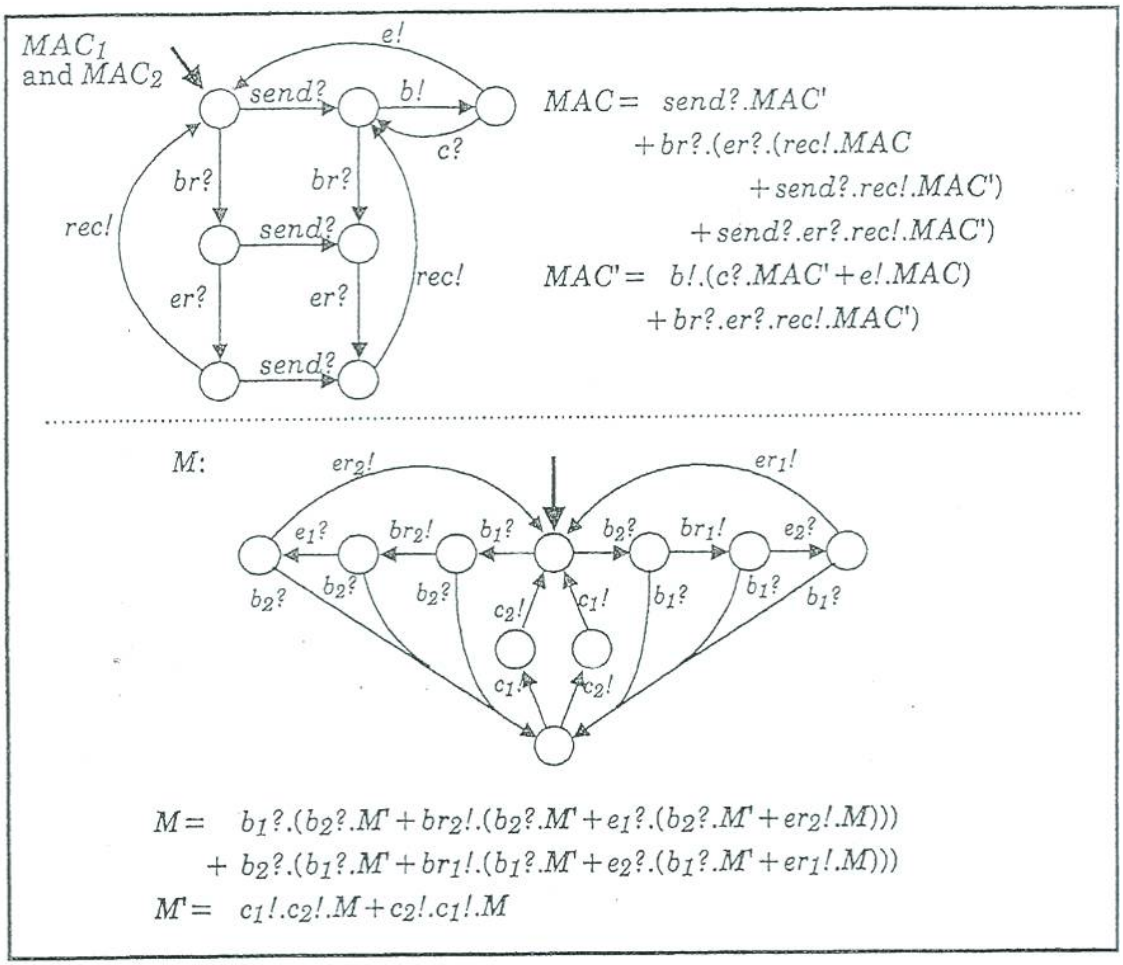

Figure 7: MAC protocol specification: entities and medium

following $b_2?$ is symmetric. Since the medium is half duplex, all of the message must be transmitted and delivered ($br_2!, e_1?, er_2!$) before a new message can be accepted. If the receiving MAC attempts transmission ($b_2?$) before all of the message has been delivered, a collision occurs. A collision results in that both MAC:s are notified of the collision ($c_1!$ and $c_2!$ in unspecified order), and that both colliding messages are lost, i.e. M returns to its original state.

2.3 Verification

Let L be the set of actions $\{b_1, br_1, e_1, er_1, c_1, b_2, br_2, e_2, er_2, c_2\}$, i.e. the actions corresponding to events in the medium. Let P (protocol specification) be defined by

$$P = (MAC_1 \mid MAC_2 \mid M) \setminus L$$

The protocol verification is to prove (recall the definition of SS from section 2.1):

$$P \approx SS$$

Using our program, this can be established automatically (the running time on a SUN 3/260 is under ten seconds). Nevertheless, it might be illuminating to see exactly why this verification result is true. From the transition diagrams corresponding to MAC_1, MAC_2, and M, a diagram corresponding to P is computed in the following way: for each state m_1 in MAC_1, m_2 in MAC_2, and m in M, construct a state $\langle m_1, m_2, m\rangle$ in P. The transitions between states are computed in accordance with the rules for CCS (hence, the simultaneous execution of two complementary actions results in the unobservable action τ). The result is presented in figure 8. The diagram corresponding to P has 35 reachable states. Transitions going from left to right and crossing a vertical thick line are $send_1$ transitions. Similarly, transitions from right to left crossing a thick line are rec_2. Transitions crossing a horizontal thick line are $send_2$ if going downwards and rec_1 if going upwards. Transitions not crossing any thick line are τ transitions. The bottom diagram additionally indicates a partitioning of the states: all states in a group (shaded box) are observation equivalent.

In the same way, we compute the transition diagram corresponding to SS (figure 9), it has 20 reachable states. The same convention for transition labels is used.

Observation equivalence between P and SS is proven by the existence of a *bisimulation* relation (cf. [4]) between the state sets in the diagrams for P and SS. A bisimulation $\mathcal{R}$ has the following property: whenever $p\mathcal{R}s$ (i.e. the states p and s are related by $\mathcal{R}$) and $p \stackrel{a}{\Longrightarrow} p'$ (i.e. p can reach p' through a sequence of transitions with the observable content a), then $s \stackrel{a}{\Longrightarrow} s'$ with $p'\mathcal{R}s'$, and symmetrically if $s \stackrel{a}{\Longrightarrow} s'$ then $p \stackrel{a}{\Longrightarrow} p'$ with $p'\mathcal{R}s'$. Intuitively, this means that each transition of p can be mimicked by a transition of s and vice versa; hence p and s can never be distinguished by an external observer.

In figures 8 and 9 the bisimulation is indicated as follows: two states are in the bisimulation relation if they are in "similar" shaded boxes (where "similar" = in approximately the same position in the diagram). In fact, SS and P are both equivalent with the simpler diagram in the bottom right part of figure 9 (12 states); this diagram is obtained by replacing each shaded box with a single state.

3 Related work

Our way to model protocols is inspired by the finite state machines with coupled transitions in [5]. The examples and figures in this paper are from [6]. There are yet few other protocol verifications in CCS or similar formalisms, examples include [7], [8], [9], and [10]. A similar automatic verification tool for CCS, and principles for optimising the analysis step, are

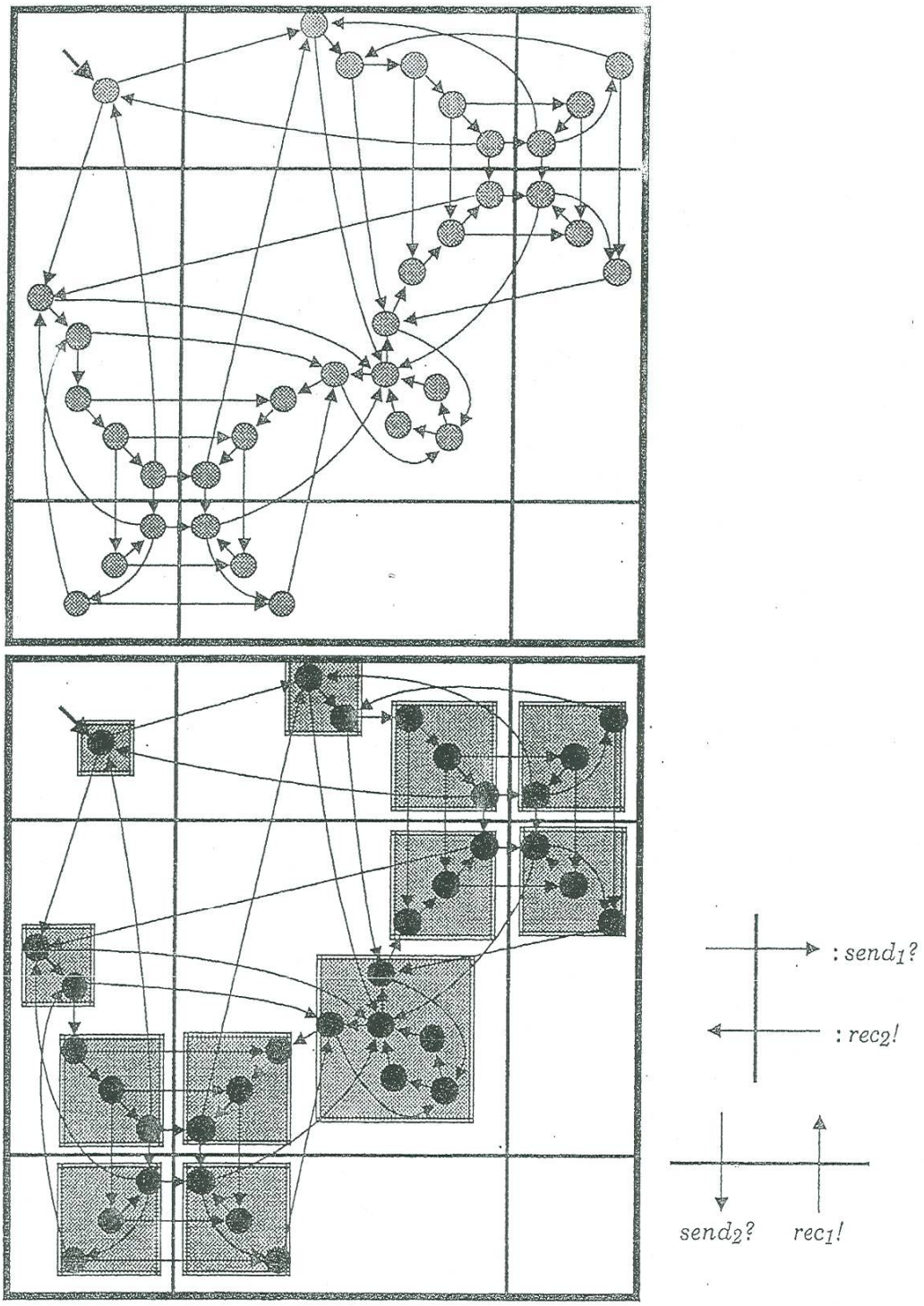

Figure 8: Transition diagram corresponding to P (top). D:o with groups of states indicated (bottom).

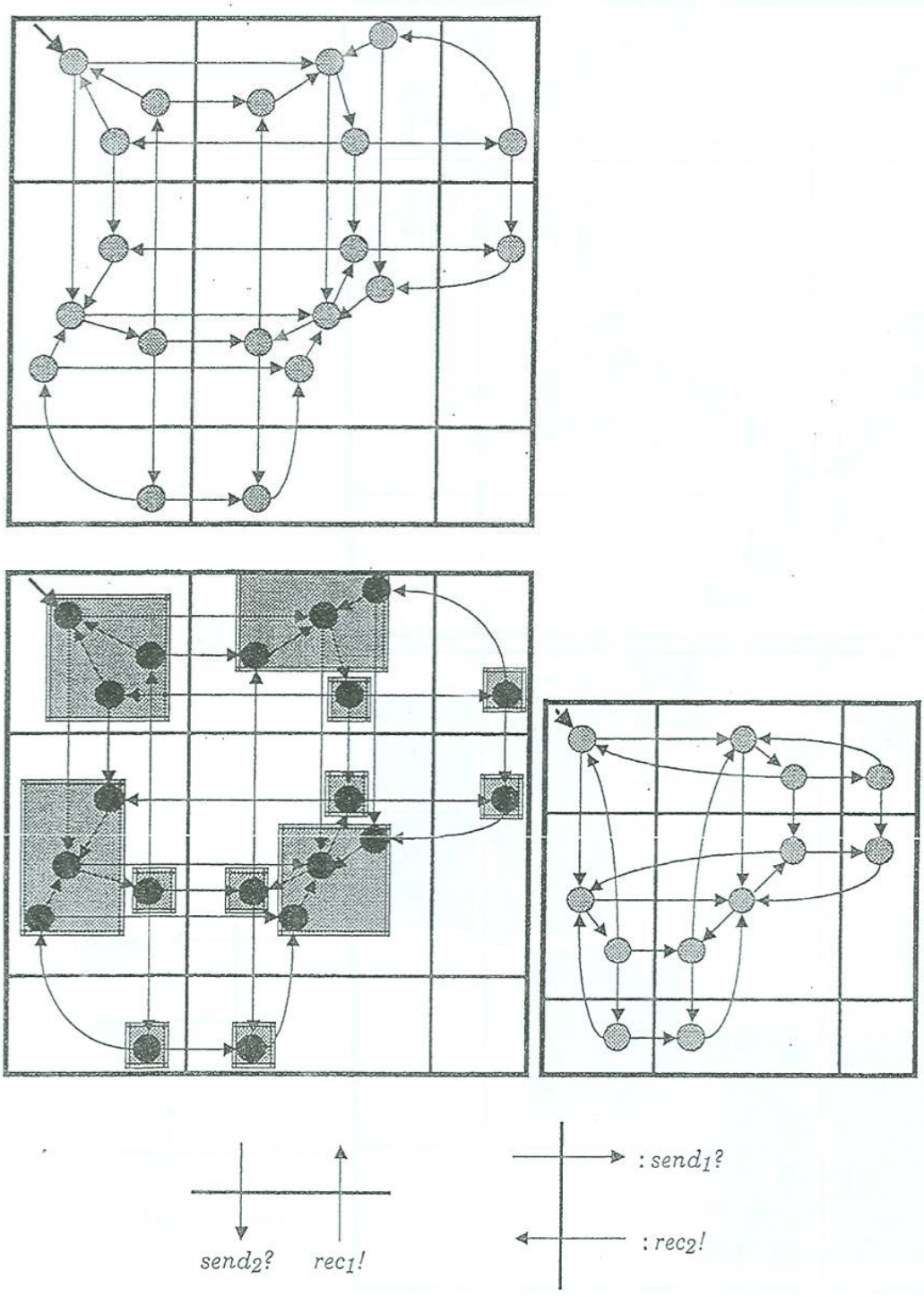

Figure 9: Transition diagram corresponding to SS (top). D:o with groups of states indicated (bottom left). A simpler equivalent diagram (bottom right).

described in [11]. CCS is used as a semantical basis for the language LOTOS, which is drafted by ISO as a protocol standard definition language ([12], [13]).

4 Discussion

Our point in this paper is that CCS can be used in medium sized examples to convey the idea of abstract behaviour. The protocol verification methodology is easy to understand and straightforward to use, but it is important to understand the limitations. Within CCS, it is impossible to describe properties related to efficiency, throughput, and other aspects related to real time. Also, it is impossible to describe fairness properties and to prove absence of "infinite chattering" or "livelocks" (but see [6] for an extension of CCS to encompass this). Observation equivalence is a very strong verification requirement: sometimes an equivalence based on tests (i.e. checking that P and SS satisfy the same tests) or traces would be sufficient. Furthermore, this use of an equivalence means that the entire service specification must be supplied in one go, and that the service must be precisely specified – it may not contain options or "don't care" clauses. Finally, CCS is not a full-fledged protocol specification language. For specifications of large protocols, a language such as LOTOS would be more appropriate.

The CCS description of the CSMA/CD protocol (first presented in [6]) is based on the Protocol Standard Definition ([3]). In order to present it in a compact way, we have made some simplifications. The actual message contents have been ignored (they do not affect the synchronisation properties of the protocol). We have only considered the case when two stations are connected, whereas the protocol is intended to be used with an arbitrary number of stations. Also, if too many collisions occur, the *MAC* may abandon retransmissions and report communication failure to its user. We have ignored this, and assumed that the service specification implies error free communication. In view of these simplifications, some service primitives in the standard (notably *MA-DATA confirm* and *carrier sense*) are irrelevant and have been omitted. We still feel that our description contains a significant part of the protocol. In any case, the same verification methodology would apply to a more detailed description of the protocol.

Our experience has shown that for the purpose of protocol verification, the service specification in the standard is incomplete. It does not specify the precise synchronisation properties of the service. Also, the protocol is heavily overspecified. The standard describes an MAC entity as two processes running in parallel and communicating through shared memory. There are also several internal timers controlling different phases of the retransmission procedure. These timers are of no importance if we are only interested in external communication events, but sometimes restrictions on time outs are formulated, implying restrictions in terms of externally observable behaviour.

The conclusion is that it is not possible to obtain a CCS description directly from the standard. We believe that the main reason for the deficiencies of the standard is that there are yet no widespread techniques for description of protocol entities in a sufficiently abstract way. Until such techniques emerge, protocol verifications are of little practical value: it is unclear how to prove the correctness of a formal description with respect to a standard definition.

Acknowledgments

I wish to thank Per Gunningberg for the use of his thesis ([14]) when writing section 1 of this paper. Many thanks to Steffen Weckner for drawing my attention to the CSMA/CD protocol, and to David Walker for a critical reading of an earlier version of this paper.

References

[1] Robin Milner. *A Calculus of Communicating Systems.* Volume 92 of *Lecture Notes of Computer Science*, Springer Verlag, 1980.

[2] S A Smolka. *Analysis of Communicating Finite State Processes.* PhD thesis, Dep. of Computer Science, Browne University, Providence, 1984. Available as report CS-84-05.

[3] *ISO Draft Standard Proposal 8802/03: Local Area Networks CSMA/CD access method and physical layer specification.* ISO/TC 97/SC 6, 1984. Also: IEEE Standard 802.3 (1983).

[4] Robin Milner. Calculi for synchrony and asynchrony. *Theoretical Computer Science*, 25:267–310, 1983.

[5] G Bochmann. Finite state descriptions of communication protocols. *Computer Networks*, 2:361–372, 1978.

[6] Joachim Parrow. *Fairness Properties in Process Algebra.* PhD thesis, Uppsala University, Uppsala, Sweden, 1985. Available as report DoCS 85/03, Department of Computer Systems, Uppsala University, Sweden.

[7] C Koomen. Algebraic specification and verification of protocols. *Science of Computer Programming*, 5(1):1–36, 1985.

[8] J Bergstra and J Klop. *Verification of an Alternating Bit Protocol by means of Process Algebra.* Technical Report, Centrum voor Wiskunde en Informatica, 1984.

[9] S.A. Smolka, A.J. Frank, and S.K. Debray. Testing protocol robustness the CCS way. In *Protocol Specification, Testing, and Verification IV (1984)*, pages 93–108, 1985. North-Holland.

[10] Kim Larsen and Robin Milner. *A Complete Protocol Verification using Relativized Bisimulation.* Technical Report ECS-LFCS-86-13, Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1986.

[11] Didier Vergamini. *Verification by means of Observation Equivalence on Automata.* Technical Report 501, INRIA, Sophia Antipolis, 1986.

[12] E Brinksma and G Karjoth. A specification of the OSI transport service in LOTOS. In *Protocol Specification, Testing, and Verification IV (1984)*, pages 227–251, 1985. North-Holland.

[13] V. Carchiolo, A. Faro, O. Mirabella, G. Pappalardo, and G. Scollo. A LOTOS specification of the PROWAY highway service. *IEEE Transactions on Computers*, C-35(11):949–968, 1986.

[14] Per Gunningberg. *Fault-Tolerance Implemented by Voting Protocols in Distributed Systems.* PhD thesis, Uppsala University, Uppsala, Sweden, 1983. Available as report UPTEC 8369R, Institute of Technology, Uppsala University, Sweden.