Projections of the Reachability Graph and Environment Models,

two Approaches to Facilitate the Functional Analysis of Systems of Cooperating Finite State Machines

Heiko Krumm
Institut für Telematik, Universität Karlsruhe
Zirkel 2, Postfach 6980, D-7500 Karlsruhe 1, West Germany

Projections of the reachability graph reduce the storage costs of the computation and support the evaluation. Secondly the specification of systems and their components is completed by restrictions to the behaviour of their environments. Components can then be analysed separately from each other, and in the analysis of a system components can be replaced by reduced substitutes. The decision of the abstract equality of indeterministic behaviours and the computation of reduced substitutes is based on the use of corresponding deterministic acceptors.

1. Introduction

For the analysis of systems of cooperating finite state machines, mostly the so-called reachability graph is computed and evaluated [6,7,8,18]. It documents all - during possible executions - reachable system states and transitions. When analysing systems modelled near to reality, problems occur from the complexity of the reachability graph. The time and storage requirements of its computation as well as the effort for its inspection and evaluation can exceed practical limits [1,9,11,12].

The first approach combines the computation of the reachability graph with the derivation of projections of it and with an integrated evaluation supporting the inspection of large graphs. The algorithm has reduced storage requirements and can be executed efficiently on a multiprocessor machine.

The second approach proposes, generally to complete each specification of a component by a so-called environment model. This contributes to the value of the specification as a starting point for the design of the component's implementation and allows one to structure the analysis. An algorithm is outlined, computing the so-called reduced substitute of the component, which mostly is much less complex than the original component, but can replace the original component in the analysis of the system.

The equivalence of components, i.e., the abstract equality of indeterministic behaviours, is decided by the comparison of languages. Thus the standard algorithms for the minimization of deterministic finite state machines can be applied.

Both approaches have been implemented in a prototypical analysis-tool running on a small workstation. It has been successfully applied during some projects at the development of distributed applications of computer networks.

In the following, chapter 2 introduces a very basic system's model, which easily can be adapted to the different constructive specification techniques (e.g., CCS[15], ESTELLE[4], LOTOS[3], Petri Nets [8]) commonly in use. Here also the characterisation of behaviours by languages is outlined. Chapter 3 describes the computation and evaluation of projections of the reachability graph. Chapter 4 concerns the concept of environment models.

2. Basic Model

Under consideration are systems, which consist of a finite set of components. To simplify the introduction of the approaches, the following assumptions are made. A system is closed, i.e., there does not exist a system's interface to an environment outside, and the communication between components always concerns exactly two parties.

2.1 Component, Behaviour and Language

Each component is defined over a finite set of component states S and a finite set of event-types $A \cup \{\varepsilon\}$, consisting of the alphabet A and a special type ε. One starting state s_0 is defined. Furthermore a transition-relation Δ is defined, consisting of triples of an actual state, an event-type and a successor-state. There may be different triples containing the same pair of actual state and event-type (indeterminism). The occurrence of ε-transitions is restricted not to form cycles (convergence).

The interpretation is as follows. In the course of time, a component performs steps, each step changing the actual state to a succesor-state in correspondence with the transition-relation and the type of events, occuring at the component's interface. The events are totally ordered in time and to each event one state-transition is performed. When for an actual state a transition labelled with ε is defined, it can take place arbitrarily without reference to an interface event (spontaneous transition). To each state a subset of $A \cup \{\varepsilon\}$ can be assigned, the ready-set (cf. [16,17]). It consists of all event-types, transitions are defined for from the state. The environment can generate an event of the alphabet of the component only, if the type of the event is in the ready-set of the actual state of the component.

To abstract from the internal representation of a component, referencing the concept of testing equivalence [16], a component shall be characterized by the set of all experiments possible with it. An experiment is a pair of words over A. The first one is called test-pattern, the second one result and is always a starting sequence of the test-pattern. The meaning is, that the environment starting an experiment at first will try to generate an event in accordance with the first character of the test-pattern. If - due to the ready-set - the event cannot be generated, the experiment stops. The result is the empty word. Otherwise the event is generated, noted down in the result, and the experiment is continued in accordance with the second character of the test-pattern, and so on, until either all events of the test-pattern have been created and noted down in the result (successful experiment) or the experiment had to be stopped due to the actual character in the test-pattern not being a member of a ready-set reachable (non-successful experiment).

To get a more concise means the set of experiments can be described by the language of the component. The alphabet of the language is A enriched by an additional special character ω. Each succesful experiment (w, w) is represented in the language by its test-pattern 'w'. Furthermore a word 'w c ω' represents the class of all non-successful experiments of the template (w c u, w). To describe the abstract behaviour of a component in an unique and minimal form, its indeterministic state-machine can be mapped into a deterministic finite state acceptor for its language, which can be minimized applying the well-known algorithms for this purpose. This is done in following steps:

- A termination state s_t, an abort state s_x, and a transition (s_x,ω,s_t) are introduced additionally. s_t and all members of S are the final states of the acceptor. The starting state still is s_0.
- For each actual state $s \in S$, for which no ε-transitions are defined, and each $c \in A$, for which no transitions (s,c,s') are defined, a transition (s,c,s_x) ist introduced, documenting that an experiment non-successfully can be stopped. Now, the indeterministic finite state machine has been transformed into a corresponding indeterministic finite state acceptor.
- By the algorithm of Myhill-Büchi the indeterministic acceptor can be transformed into a corresponding deterministic one.

2.2 System

The formal semantics of a system is its reachability graph. A node corresponds to a system state and is a vector of the different actual component states. A directed edge corresponds to a system step caused by the occurence of exactly one event. The graph documents all the system states and transitions which can occur in the possible executions of the system.

2.3 Example

Fig. 1 shows a system acting in accordance to a simplification of the well-known Alternating-Bit-Protocol [2]. The two components S and E are connected by an unreliable medium D' and provide to the two user-components L and R a reliable communication service. A component is specified by its transition list, the starting state is labelled by 0. The last two sections describe the reachability graph (the ordering of the system state components is L, R, S, E, D').

component L
(0,rq,1), (1,cn,0)

component R
(0,in,1), (1,r2,0)

component S
(0,rq,1), (1,m0,2), (2,q1,1), (2,qg,1), (2,q0,3), (3,cn,4), (4,rq,5), (5,m1,6), (6,q0,5), (6,qg,5), (6,q1,7),(7,cn,0)

component E
(0,n0,1), (1,in,2), (2,re,3), (3,p0,4), (4,n0,3), (4,ng,3), (4,n1,5), (5,in,6), (6,re,7), (7,p1,0), (0,n1,7), (0,ng,7)

component D'
(0,m0,1), (0,m0,2), (0,m1,2), (0,m1,3), (1,n0,4), (2,ng,4), (3,n1,4), (4,p0,5), (4,p0,6), (4,p1,6), (4,p1,7), (5,q0,0), (6,qg,0), (7,q1,0)

System States
0:(0,0,0,0,0), 1:(1,0,1,0,0), 2:(1,0,2,0,2), 3:(1,0,2,7,4), 4:(1,0,2,0,7), 5:(1,0,2,0,6), 6:(1,0,2,0,1), 7:(1,0,2,1,4), 8:(1,1,2,2,4), 9:(1,0,2,3,4), 10:(1,0,2,4,6), 11:(1,0,1,4,0), 12:(1,0,2,4,1), 13:(1,0,2,4,2), 14:(1,0,2,4,5), 15:(1,0,3,4,0), 16:(0,0,4,4,0), 17:(1,0,5,4,0), 18:(1,0,6,4,2), 19:(1,0,6,3,4), 20:(1,0,6,4,5), 21:(1,0,6,4,6), 22:(1,0,6,4,3), 23:(1,0,6,5,4), 24:(1,1,6,6,4), 25:(1,0,6,7,4), 26:(1,0,6,0,6), 27:(1,0,5,0,0), 28:(1,0,6,0,3), 29:(1,0,6,0,2), 30:(1,0,6,0,7), 31:(1,0,7,0,0)

System Transitions
(0,rq,1), (1,m0,2), (2,ng,3), (3,p1,4), (4,q1,1), (3,p1,5), (5,qg,1), (1,m0,6), (6,n0,7), (7,in,8), (8,re,9), (9,p0,10), (10,qg,11), (11,m0,12), (12,n0,9), (11,m0,13), (13,ng,9), (9,p0,14), (14,q0,15), (15,cn,16), (16,rq,17), (17,m1,18), (18,ng,19), (19,p0,20), (20,q0,17), (19,p0,21), (21,qg,17), (17,m1,22), (22,n1,23), (23,in,24), (24,re,25), (25,p1,26), (26,qg,27), (27,m1,28), (28,n1,25), (27,m1,29), (29,ng,25), (25,p1,30), (30,q1,31), (31,cn,0)

Fig. 1: Example System

3. Projections and Integrated Evaluation

Instead of computing the complete reachability graph only a projection of it is computed. A projection is defined by a scope, which is a subset of the alphabet of the system. The graph is called topgraph and contains the system starting state and moreover only such states, which are reached directly after the occurence of an event of the scope. To keep the analysis exhaustive, intermediate nodes and transitions are computed and represented in so-called subgraphs. To a subgraph one node of the topgraph serves as starting state. Moreover, it only contains the system states reachable under the occurence of events of the complement of the scope.

Different subgraphs can be computed simultaneously and independently from each other. Thus at one hand multiprocessor machines can be used efficiently. At the other hand, after the computation and evaluation of a subgraph, the subgraph can be deleted from memory. Therefore the memory requirements can be reduced in comparison with the computation of the complete reachability graph. To document properties of the subgraphs which are relevant for the system's analysis, the nodes of the topgraph are attributed by corresponding flags.

3.1 Example

Fig. 2 shows the topgraph and the subgraphs of the system of Fig. 1 for the scope {rq, cn, m0, m1, q0, q1, qg}. Because the subgraphs only concern the complement of the scope, all of those components can be ignored, the alphabet of which is fully contained in the scope.

Topgraph Nodes
0:(0,0,0,0,0), 1:(1,0,1,0,0), 2:(1,0,2,0,2), 6:(1,0,2,0,1), 11:(1,0,1,4,0), 12:(1,0,2,4,1), 13:(1,0,2,4,2), 15:(1,0,3,4,0), 16:(0,0,4,4,0), 17:(1,0,5,4,0), 18:(1,0,6,4,2), 22:(1,0,6,4,3), 27:(1,0,5,0,0), 28:(1,0,6,0,3), 29:(1,0,6,0,2), 31:(1,0,7,0,0)
Topgraph Transitions
(0,rq,1), (1,m0,2),(2,q1,1), (2,qg,1), (1,m0,6), (6,qg,11), (11,m0,12), (11,m0,13), (12,qg,11), (13,qg,11), (6,q0,15), (12,q0,15), (13,q0,15), (15,cn,16), (16,rq,17), (17,m1,18), (18,q0,17),(18,qg,17), (17,m1,22), (22,qg,27), (27,m1,28),(27,m1,29), (28,qg,27), (29,qg,27), (22,q1,31), (28,q1,31), (29,q1,31), (31,cn,0)

Subgraph A
((-,0,-,0,2),ng,(-,0,-,7,4)), ((-,0,-,7,4),p1,(-,0,-,0,6)), ((-,0,-,7,4),p1,(-,0,-,0,7))
Subgraph B
((-,0,-,4,1),n0,(-,0,-,3,4)), ((-,0,-,3,4),p0,(-,0,-,4,6)), ((-,0,-,3,4),p0,(-,0,-,4,5))
Subgraph C
((-,0,-,0,1),n0,(-,0,-,1,4)), ((-,0,-,1,4),in,(-,1,-,2,4)), ((-,1,-,2,4),re,(-,0,-,3,4)), ((-,0,-,3,4),p0,(-,0,-,4,5)), ((-,0,-,3,4),p0,(-,0,-,4,6))
Subgraph D
((-,0,-,4,3),n1,(-,0,-,5,4)), ((-,0,-,5,4),in,(-,1,-,6,4)), ((-,1,-,6,4),re,(-,0,-,7,4)), ((-,0,-,7,4),p1,(-,0,-,0,6)), ((-,0,-,7,4),p1,(-,0,-,0,7))
Subgraph E
((-,0,-,0,3),n1,(-,0,-,7,4)), ((-,0,-,7,4),p1,(-,0,-,0,6)), ((-,0,-,7,4),p1,(-,0,-,0,7))
Subgraph F
((-,0,-,4,2),ng,(-,0,-,3,4)), ((-,0,-,3,4),p0,(-,0,-,4,5)), ((-,0,-,3,4),p0,(-,0,-,4,6))

Fig. 2: Projection to {rq, cn, m0, m1, q0, q1, qg}

The storage requirements are reduced now. Instead of 32 system states, the topgraph only contains 16. The largest subgraph contains 6 nodes restricted to 3 components. Thus roughly a reduction of (32-16-6*3/5)/32 ≈ 40% has been possible. With real examples and scopes of interest reductions between 40% and 60% have been possible.

The time requirements are increased now, since some system states will be computed more than once. In the prototypical implementation therefore subgraphs are deleted from memory only, when no more storage is available. Then an acceleration is possible, since - due to the component restriction of subgraphs - some subgraphs can be used more than once.

With respect to the execution of the algorithm on a multiprocessor machine, in the example the use of two processors computing subgraphs in parallel will accelerate the computation. Roughly, instead of 16 time units by use of one processor, the computation can be done in 9 time units, if each subgraph computation can be executed in one unit of time.

Variants of the algorithm studied also in the prototypical implementation use more than two graph levels. A subgraph can reference to subsequent subgraphs, too, if the complement of the scope is structured further. Since this structuring can be chosen with respect to the optimization of the number of components of subgraph states, further reductions of the storage requirements are possible. At practical examples we get space reductions up to 85%.

3.2 Evaluation

Although the result of the computation is a projection, all system states reachable and all transitions executable have been computed. Therefore the general criterium 'All specification constructs have to be of relevance for a system' can be decided easily by marking the constructs during the computation. Moreover each subgraph is evaluated after its computation:

- Are there termination nodes, from which no transition is possible at all?
- Are there cycles within the subgraph? A cycle, which can be left by an event of the scope already in its first node is classified as independent. A cycle, which never can be left by events of the scope, is called quasi-terminating. Cycles, wich can be left after some steps are called delaying.

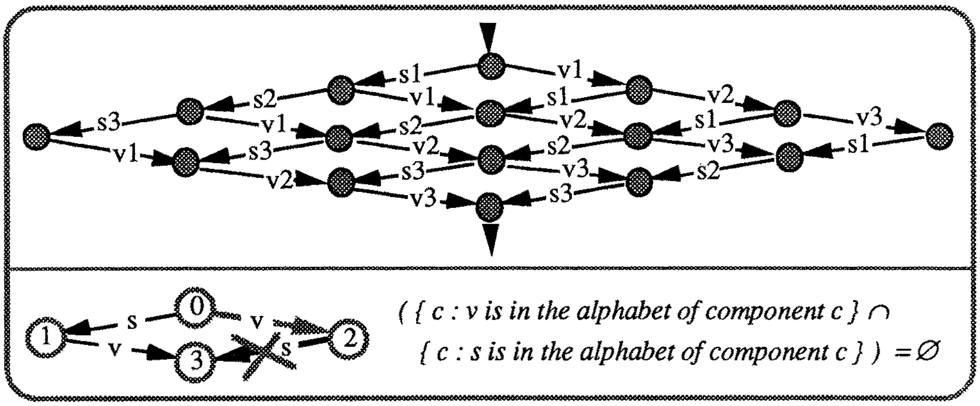
Thereafter that state of the topgraph, which has opened the subgraph, is attributed by the set of respective flags.

At the analysis of computer network communication protocols this has facilitated the evaluation of projections by human inspection enormously. In different projects, all the design errors detected, had been detected by the use of small projections and human inspection in consideration of the flags. Besides of the delay and independency flags all others had signaled design errors. The delay flag very often occured, signaling that the protocol progress may be delayed forever, when the medium will disturb each further transfer. Then we added to the scope some event-types of the alphabet of the medium to analyse the cycles in more detail. Independent cycles occur in connection with time-out mechanisms.

Topgraph Nodes
0:(0,0,0,0,0), 1:(1,0,1,0,0) delay,8:(1,1,2,2,4), 9:(1,0,2,3,4) delay, 16:(0,0,4,4,0), 17:(1,0,5,4,0) delay, 24:(1,1,6,6,4), 25:(1,0,6,7,4) delay
Topgraph Transitions
(0,rq,1), (1,in,8), (8,re,9), (9,cn,16), (16,rq,17), (17,in,24), (24,re,25), (25,cn,0)

Fig. 3: Projection to {rq, cn, in, re}

Fig. 3 shows the projection of the example system with respect to the scope of the event-types of the service to be provided. There are four nodes, in which the progress can be delayed, if the medium will disturb the transfer of messages forever. To abstract from this possible divergence of the behaviour at the service-interface, additional assumptions based on performance considerations have to be introduced. Projections, the scope of which also contains event-types of the medium-interface, can help to prepare these considerations.



({ *c : v is in the alphabet of component c* } ∩ { *c : s is in the alphabet of component c* }) = ∅

Fig.4: Concurrency Pattern

3.3 Concurrency Patterns

If there are several concurrently running event-sequences possible in a system, the reachability graph will document this by concurrency patterns. Fig. 4 shows such a pattern for the two event sequences (s1, s2, s3) and (v1, v2, v3). Let the event-types of the first sequence be in the scope of a projection and those of the second outside the scope. Now the algorithm can work very inefficiently, since 12 nodes open a subgraph and from each subgraph-node a transition to a node of the topgraph is possible.

The lower part of Fig. 4 shows, how this can be avoided under certain, but oftenly appearing circumstances. The computation of a transition from a subgraph node to a topgraph node can be skipped, if the resulting topgraph node will be computed also due to the tracing of a predecessor of the node (at cycles one edge has to be ommitted in this consideration). If in the example the condition is true for all pairs of members of the two event sequences only 4 nodes of the topgraph and only 4 subgraphs have to be computed. At the variant of the algorithm, at which more than two graph-levels are used, the scopes of deeper levels are chosen in a way, that the condition always is true for event types of the scopes of graphs, the levels of which differ by two or more.

4. Environment Models

The additional introduction of an environment-description into the specification of a component states behavioural restrictions, each environment of the component is assumed to meet. Thus it provides for the possibility of also noting down the designer's intentions about the general properties of the environment as they appear to the component. This completes the specification, moreover, it renders possible to decouple the analysis of the components of complex systems, i.e., to analyse a subsystem separately from its specific environment and to analyse a specific environment with respect to its conformance with the subsystem's environment-conditions. When the specification is used as starting point for an implementation, the environment-description supports the design of efficient solutions, which are tailored to the restrictions noted down there.

In the area of sequential programs the environment-model corresponds to the precondition of a specification [10]. In the specification of concurrent systems by operational models one commonly concentrates on the component itself and few approaches only propose the explicit notion of environment conditions (e.g. [5,13,14]).

```
component S                 environment of S
 (0,rq,1), (1,m0,2),         user model
 (2,q1,1), (2,qg,1),          (0,rq,1), (1,cn,0)
 (2,q0,3), (3,cn,4),
 (4,rq,5), (5,m1,6),         medium model
 (6,q0,5), (6,qg,5),          (0,m0,1), (0,m0,2), (0, m0,3), (0, m1,1), (0, m1,2),
 (6,q1,7),(7,cn,0)            (0, m1,3), (1, q0, 0), (2,qg,0), (3,q1,0)
environment ready-sets of S
 0:{rq,m0,m1}, 1:{cn,m0,m1}, 2:{cn,q0,q1,qg}, 3:{cn,m0,m1}, 4:{rq,m0,m1},
 5:{cn,m0,m1}, 6:{cn,q0,q1,qg}, 7:{cn,m0,m1}
reduced substitute of S
 (0',cn,0'), (0',rq,1'), (1',m0,1'), (1',qg,1'), (1',q1,1'), (1',q0,2'),
 (2',cn,2'), (2',rq,3'), (3',m1,3'), (3',qg,3'), (3',q0,3'), (3',q1,0')
```

Fig. 5: Environment Model and Reduced Substitute of S

4.1 Example

Fig. 5 shows in the upper section the specification of the component S of the example of Fig. 1. It contains a model of the environment, namely two concurrently running machines representing the general left user and the general left interface to the medium. The medium as it is assumed to appear to S is modeled in a very general way by means of indeterministic transitions. Although the specification of S is minimal, with respect to the environment model it can be reduced further. The reduced substitute shown in the lower part only contains 4 states instead of 8, and at the analysis of the whole system the number of combinatorially possible states is divided by two, facilitating the efficient representation of graph nodes.

4.2 Reduced Substitute

To compute the reduced substitute following steps are performed:

- The reachability graph of the specification, i.e., of the small system consisting of the component S and its environment-model, is computed. The environment ready-set of a graph node is the union of the actual ready-sets of the different evironment components.
- The environment ready-set of a state s of S is $U \cap A$, where U is the union of the environment ready-sets of all nodes, which contain s as actual S-component in the state vector, and A is the alphabet of S.
- In accordance with chapter 2 the deterministic acceptor for the component S is built, while the environment ready-set of a state is respected. Transitions (s,c,s_x) are introduced for such members c of the alphabet only, which are in the environment ready-set of s. Now the language of the acceptor documents those experiments only, which can be performed by the restricted environment.
- The acceptor can be minimized. Additionally two inequivalent states can be merged, if they are equivalent only with respect to the characters for which at both states transitions are defined.

A reduced substitute can replace an original component in the analysis of a system, if the consistency of the system with respect to the environment model is given.

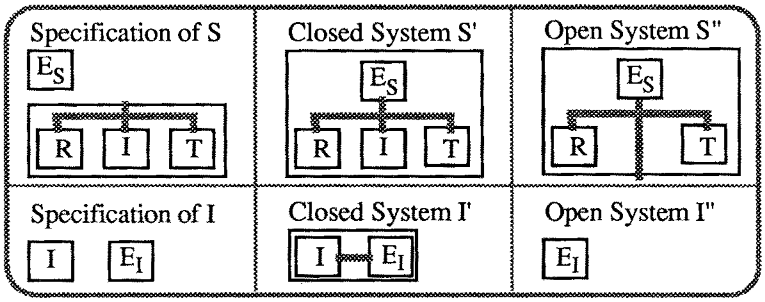


Fig. 6: System Structures for the Environment Consistency Check

4.3 Environment Consistency

To analyse, whether a given system meets the restrictions of the environment model of a component, one has to check, whether the set of experiments, the system can perform with a component, is a subset of the experiments, the environment model can perform. Fig. 6 shows the system structures used in the explanation of the two following criteria. We assume, that the reachability graph of an open system can be computed, and a single finite state machine representation of its interface behaviour, the general substitute, can be derived as well as its language.

A sufficient condition for the environment consistency is, that the language of S" is a subset of the language of I". Then in S' only those experiments with I will be possible, which also are possible in I'. To decide the consistency the reachability graphs of the systems S' and I' can be computed, and the two respective environment restricted acceptors of I can be deduced. The corresponding languages exactly describe the sets of experiments possible in the two environments and can be compared with each other.

5. Conclusion

During the design of more complex systems of practical interest, i.e., distributed applications of computer networks (e.g., Remote Operation Service, Filing-Service, File-Transfer), the methods described have been applied. Their application has been supported by a respective analysis tool, which performs the computation of projections and languages and the derivation of reduced substitutes. The tool runs on a small workstation (2 Mbyte, 68020 CPU) and therefore storage and computation time limitations heavily forced the modelling under abstractions and the structuring of the analysis by defining subsystems, analysing them separately, and replacing them by their reduced substitutes.

However, especially the structuring and the respective different checks of the environment consistency turned out to be very useful. They supported the insight in and the understanding of the mechanisms under development. Thus, design errors have been detected very early. Furthermore, the sum of the analysis results has given helpful hints when designing the implementations. Especially the reduced substitute of a component can be valued as a very compact starting point for the design of an efficient implementation.

References

[1] Aggarwal, S.; Barbara, D.; Meth, K.; SPANNER: A Tool for the Specification, Analysis, and Evaluation of Protocols; IEEE Transactions on Software Engineering 13,12(1987)1218-1237

[2] Bartlett, K.; Scantlebury, R.; Wilkonson, P.; A Note on Reliable Full Duplex Transmissions over Half-Duplex Links; Communications of the ACM 12,5(1969)260-261

[3] Bolognesi, T.; Brinksma, E.; Introduction into the ISO Specification Language LOTOS; Computer Networks and ISDN Systems 14(1987)25-59

[4] Budkowski, S.; Dembinski, P.; An Introduction to Estelle: A Specification Language for Distributed Systems; Computer Networks and ISDN Systems 14(1987)3-23

[5] Burkhardt, H.; Eckert, H.; Prinoth, R.; Modelling of OSI-Communication Services and Protocols Using Predicate/Transition Nets; in Protocol Specification, Testing, and Verification IV, Y. Yemini, R. Strom, S. Yemini (eds.), North-Holland, Amsterdam (1985)165-192

[6] Cavalli, A.; Paul, E.; Exhaustive Analysis and Simulation for Distributed Systems, Both Sides of the Same Coin; Distributed Computing 2(1988)213-225

[7] Danthine, A.; Protocol Representation with Finite State Models; in P. Green (ed.), Computer Network Architectures and Protocols, Plenum Press, New York (1983)579-606

[8] Diaz, M.; Modeling and Analysis of Communication and Cooperation Protocols using Petri-Net Based Models; Computer Networks, 6(1982)419-441

[9] Fernandez, J.; Richier, J.; Voiron, J.; Verification of Protocol Specifications Using the Cesar System; in M. Diaz (ed.), Protocol Specification, Testing, and Verification V, North-Holland, Amsterdam (1986)71-90

[10] Hoare, C.; An Axiomatic Basis for Computer Programming; Communications of the ACM 12(1969)576-580

[11] Holzmann, G.; Automated Protocol Validation in Argos: Assertion Proving and Scatter Searching; IEEE Transactions on Software Engineering 13,6(1987)683-696

[12] Holzmann, G.; On Limits and Possibilities of Automated Protocol Analysis; in Protocol Specification, Testing, and Verification VII, H. Rudin, C. West (eds.), North-Holland, Amsterdam (1987)339-344

[13] Krumm, H.; Drobnik, O.; Interactive Verification of Communication Software on the Basis of CIL; Computer Communications Review 14,2(1984)92-99

[14] Krumm, H.; Drobnik, O.; Problem-Oriented Logical Specifications of Commmunication Services and Protocols; in Proc. of the 8th International Conference on Computer Communication(ICCC), P. Kühn (ed.), Elsevier Science Publishers, (1986)474-478

[15] Milner, R.; A Calculus of Communicating Systems (CCS); Springer Verlag, Berlin (1980)

[16] Nicola de, R.; Hennessy, M.; Testing Equivalences for Processes; Theoretical Computer Science 34 (1984)83-133

[17] Olderog, E.-R.; Hoare, C.; Specification-Oriented Semantics for Communicating Processes; Acta Informatica 23(1986)9-66

[18] Rudin,H.; Tools for Protocols Driven by Formal Specifications; in Kündig, A.; Bührer, R.; Dähler,J.(eds.); Embedded Systems; Springer Verlag, Berlin (1987)127-152