Automated Verification of Communication Protocols using CCS and BDDs

Reiner Lichtenecker[1], Klaus Gotthardt[1] and Janusz Zalewski[2]

[1] FernUniversität-Hagen, Dept. of Electr. Engineering, D-58084-Hagen, Germany
[2] University of Central Florida, Dept. of Electr. and Comp. Engineering, Orlando, FL 32812, USA

Abstract. The application of formal methods in protocol verification is of great importance, especially in the area of safety critical systems. Formal methods, however, are scarcely used in industrial practice today because they are hardly to integrate into the conventional system design and require a high effort in computing. We describe the implementation and application of a tool that handles formal specifications written in the process calculus CCS. The automatic verification process is based on binary decision diagrams to efficiently cope with state explosion problems. As an verification example we use a model of the CSMA/CD protocol including propagation delay effects on the transmission medium.

1 Introduction

The requirements in software development increasingly evolve towards provably correct software design, especially in safety-critical applications. A precondition for this is a systematic software development process, which guarantees adherence of the application to specified properties already during design. The objective of formal verification is to establish this proof by an abstract implementation model. A number of formalisms for system modeling and verification are known from the literature. Among these formalisms we especially consider the algebraic process calculus CCS [1]. In CCS correctness is proven by establishing a bisimulation equivalence between the implementation model and a reference specification. This task can be supported by suitable software tools, such as the well known *Concurrency Workbench* (CWB) [3]. However, automated proofs often fail due to the complexity of real systems which lead to huge state spaces. This can already happen in the first step: the generation of the labelled transition system from the CCS specification.

Since their introduction by Bryant [5], Binary Decisions Diagrams (BDDs) have led to a substantial increase of the state space size of automatically verifiable systems [6] [7], especially in the area of hardware verification. Thus, application of this technique to the verification of process algebraic specifications is an obvious idea. A basic discussion of this approach can be found in [8]. However, automatic translation from CCS models to BDDs and the application to models of realistic systems have not been investigated by now. We describe first experiences in the design and application of a tool for automatic verification

of CCS models with BDDs. We present the implementation of CCS operators like composition or summation and proofs by bisimulation. As application examples we consider the model of a scheduler [1] and a realistic model of the CSMA/CD-protocol.

2 The process calculus CCS

In CCS a process is described by its interaction with other processes and the environment. Its behaviour is defined as the possible sequences of atomic actions it can execute. The operators of CCS are shown in (1).

$$E ::= \alpha.E \mid \mathbf{0} \mid E_1 + E_2 \mid E_1 \mid E_2 \mid A \stackrel{\text{def}}{=} E_i \mid E[f] \mid P\backslash L \tag{1}$$

For a comprehensive explanation of CCS, the reader is referred to [1]. One way of verifying the desired behaviour of an implementation is to prove an equivalence between its CCS specification and a reference model. Among the many possible equivalence relations, (weak) bisimulation is probably the most useful, as it abstracts from the internal τ-actions and is still capable of detecting deadlocks. Bisimulation equivalence of two processes P and Q is defined as follows:

A binary relation $\mathcal{S} \subseteq \mathcal{P} \times \mathcal{P}$ over processes is a *bisimulation*, if $(P, Q) \in \mathcal{S}$ implies for all actions from the action set $(\alpha \in Act)$

$$\begin{array}{ll} (i) & \text{Whenever } P \xrightarrow{\alpha} P', \text{ then, for some } Q', Q \stackrel{\hat{\alpha}}{\Longrightarrow} Q' \text{ and } (P', Q') \in \mathcal{S} \\ (ii) & \text{Whenever } Q \xrightarrow{\alpha} Q', \text{ then, for some } P', P \stackrel{\hat{\alpha}}{\Longrightarrow} P' \text{ und } (P', Q') \in \mathcal{S} \end{array} \tag{2}$$

$P \stackrel{\hat{\alpha}}{\Longrightarrow} P'$ denotes an observable action α with any number of preceding or succeeding τ-actions.

3 Binary Decision Diagrams (BDDs)

BDDs are a canonical representation of boolean functions. A BDD is an acyclic, directed graph which codes a binary decision tree. The variables of the BDD are totally ordered, so that along every path from the root to the leafes the variables always occur in the same order. Fig. 1 illustrates the creation of a BDD from the decision tree of the function $(a \vee b) \wedge c$ for the variable ordering $a \lhd b \lhd c$.

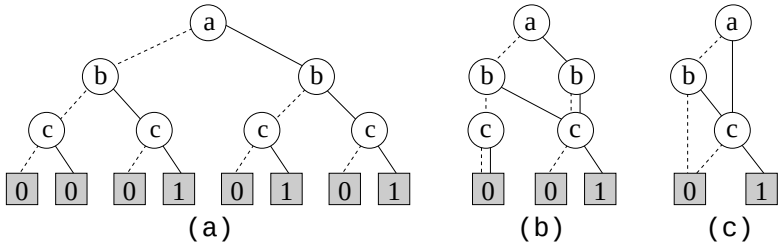


Fig. 1. Building a BDD of the function $(a \vee b) \wedge c$

Every node of the tree is labelled by a variable and refers to two subtrees, which are called *Lo* and *Hi*. Values 0 and 1 represent the leaves of the tree.

The solid edges refer to the subtree Hi which will be reached, if the value 1 is assigned to the variable. Correspondingly the dashed lines lead to the subtree Lo which will be reached if 0 is assigned. An efficient representation of the function is achieved by merging identical subtrees, as shown in Fig. 1(b). Furthermore, nodes with identical subtrees Lo and Hi will be removed from the graph. By repeatedly applying these rules we finally arrive at the BDD shown in Fig. 1(c).

3.1 Operations on Binary Decision Diagrams

A basic operation on BDDs is the *restriction* $f\mid_{x=v}$, which assigns a constant value $v \in \{0,1\}$ to variable x in function f. The restriction can be efficiently computed in the case that x is the variable of the root node of this graph:

$$f\mid x=v \stackrel{\text{def}}{=} \begin{cases} Lo(f) & \text{if } v=0 \\ Hi(f) & \text{if } v=1 \end{cases}$$

According to the Shannon expansion any binary operation $\langle \text{op} \rangle$ on BDDs can be evaluated using the following recursive definition:

$$f \langle \text{op} \rangle g \equiv [\neg x \wedge (f\mid_{x=0} \langle \text{op} \rangle g\mid_{x=0})] \vee [x \wedge (f\mid_{x=1} \langle \text{op} \rangle g\mid_{x=1})]$$

Efficient computation algorithms are given by Bryant [5]. In average the complexity of binary operations is in $O(|f| \cdot |g|)$, where $|f|$ denotes the number of nodes in BDD f. In addition to the combinatorial operators, existential quantification, universal quantification and renaming of variables will be needed later. Using the restriction operator, $\exists x.f$ and $\forall x.f$ evaluate to:

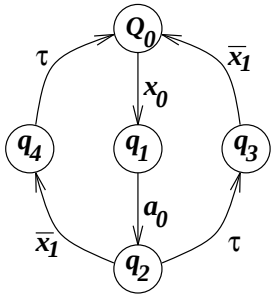
$$\exists x.f = f\mid_{x=0} \vee f\mid_{x=1} \qquad\qquad \forall x.f = f\mid_{x=0} \wedge f\mid_{x=1}$$

Substitution of a variable x by another variable y in a BDD f, denoted by $f[y/x]$, can be done based on existential quantification: $f[y/x] \equiv \exists x.(x=y) \wedge f$
Operations like restriction and existential quantification extend to vectors of binary variables in the obvious way. If $x=\{x_0, x_1, \ldots x_{n-1}\}$ is such a vector and $v \in \{0,1\}^n$ is a vector with constant elements, then we have:

$$f\mid x=v \equiv f\mid x_0=v_0 \mid x_1=v_1 \mid x_2=v_2 \ \cdots \mid x_{n-1}=v_{n-1}$$
$$\exists x.f \equiv \exists x_0.\exists x_1.\ldots.\exists x_{n-1}.f$$

3.2 Evaluation of CCS operators

CCS specifications can always be represented as labelled transition systems. With BDDs we describe such a system as a relation $R(\alpha, x, x')$ where α represents a CCS action and the states x, x' correspond to processes of the CCS specification. The relation R is a binary function which can be represented by BDDs in a straightforward way. If the transition $x \xrightarrow{\alpha} x'$ denoted by the triple (α, x, x') is an element of the relation, i.e. $R(\alpha, x, x') = 1$, then the transition system includes a state or process x which will pass over into the process x' by an action α. As an example Fig. 2 illustrates the relation of a CCS specification, labelled transition system and transition relation. It should be observed that process Q_0 is not represented by the whole transition system, but rather by the



CCS representation:

$$Q_0 = x_0.a_0.(\tau.\bar{x}_1.Q_0 + \bar{x}_1.\tau.Q_0)$$

representation as transition relation:

$$\begin{aligned} R_{Q_0}(\alpha, x, x') = {} & (\alpha = x_0) \wedge (x = Q_0) \wedge (x' = q_1) \\ & \vee (\alpha = a_0) \wedge (x = q_1) \wedge (x' = q_2) \\ & \vee (\alpha = \tau) \wedge (x = q_2) \wedge (x' = q_3) \\ & \vee (\alpha = \bar{x}_1) \wedge (x = q_3) \wedge (x' = Q_0) \vee \ldots \end{aligned}$$

Fig. 2. representation of a process in CCS and as transition relation

initial state which is labelled Q_0. States q_i do not explicitly appear in the CCS specification.

To enable the evaluation of CCS operators like *restriction*, *composition* and *relabelling* on transition systems represented by BDDs, we first have to define an encoding of actions and states. The encoding of states in elementary transition systems, i.e. systems only described using prefix- and sum-operators in CCS, is apparent: if the system consists of M states, they will be encoded by the binary representation of numbers 0 to $(M-1)$. The state vector $x = \{x_1, x_2, \ldots\}$ consists of $\lceil \mathrm{ld}(M) \rceil$ binary valued variables. The set of all occuring actions α can be represented in a similar way. However, we have to account for the fact that the τ-action and complementary actions should be easily recognizable in order to minimize the computational effort. We therefore always assign the value 0 to the τ-action. Complementary actions are distinguished by the first Bit of the vector which encodes an action α. We mark an output-action by the value $\alpha_0 = 0$ and the complementary input-action by $\alpha_0 = 1$.

Composition: The composition of two systems $R_1(\alpha, x_1, x_1')$ and $R_2(\alpha, x_2, x_2')$ generates a new system $R_p(\alpha, \langle x_1, x_2 \rangle, \langle x_1', x_2' \rangle)$. The transition relation of system R_p consists of several subrelations. In state $\langle x_1, x_2 \rangle$ system R_p can execute every transition which R_1 is able to execute in state x_1. Hence, the composition contains all possible transitions defined by the set $R_1(\alpha, x_1, x_1') \wedge (x_2' = x_2)$. Likewise, the system can perform any transition from subsystem R_2 which is characterized by $R_2(\alpha, x_2, x_2') \wedge (x_1' = x_1)$. Finally we have to consider the set of τ-transitions resulting from a simultaneous state transition of both subsystems by complementary actions. Due to the chosen encoding of actions, where α_0 distinguishes complementary actions, $\exists\, \alpha.(R_1|_{\alpha_0=0} \wedge R_2|_{\alpha_0=1})$ specifies the set of all state transitions in which R_1 executes an output and subsystem R_2 the corresponding input. It follows for the relation R_P:

$$\begin{aligned} R_p(\alpha, \langle x_1, x_2 \rangle, \langle x_1', x_2' \rangle) &= R_1(\alpha, x_1, x_1') \mid R_2(\alpha, x_2, x_2') \\ &= R_1(\alpha, x_1, x_1') \wedge (x_2' = x_2) \vee R_2(\alpha, x_2, x_2') \wedge (x_1' = x_1) \\ &\vee (\alpha = \tau) \wedge \exists\, \alpha.[(R_1|_{\alpha_0=0} \wedge R_2|_{\alpha_0=1}) \vee (R_1|_{\alpha_0=1} \wedge R_2|_{\alpha_0=0})] \end{aligned}$$

Restriction: $R(\alpha, x, x') \backslash [a_1, a_2, \ldots]$ removes all transitions from the relation caused by actions $[a_1, a_2, \ldots]$ and co-actions $[\bar{a}_1, \bar{a}_2, \ldots]$. The set of removed actions is represented as a BDD. As the new transition relation we have:

$$R(\alpha, x, x') \backslash [a_1, a_2, \ldots] = R(\alpha, x, x') \wedge \neg \exists\, \alpha_0\, .[(\alpha = a_1) \vee (\alpha = a_2) \vee \ldots]$$

With the existential quantification of α_0, transitions by actions $[a_1, ..., \bar{a}_1, ...]$ will be removed according to the definition of restriction.

Relabelling: For the *relabelling* of an action we have:

$$\begin{aligned} R(\alpha, x, x')[b/a] = R(\alpha, x, x') \wedge (\alpha \neq a) \wedge (\alpha \neq \overline{a}) \\ \vee\ (\alpha = b) \wedge R(\alpha, x, x')\ |_{\alpha = a} \vee (\alpha = \overline{b}) \wedge R(\alpha, x, x')\ |_{\alpha = \overline{a}} \end{aligned}$$

Restriction $R(\alpha, x, x')|_{\alpha=a}$ yields the set of transitions $x \xrightarrow{a} x'$ of the system caused by an input-action a.

3.3 Bisimulation Proof

The definition of bisimulation in eq. (2) can be translated into a computation on BDDs. The transition relation $\xrightarrow{\alpha}$ corresponds to the representation of the relation $R(\alpha, x, x')$. The representation of the relation $\stackrel{\hat{\alpha}}{\Longrightarrow} = \tau^* . \alpha . \tau^*$ contains additional τ-transitions $(\alpha = \tau) \wedge (x' = x)$ connecting every state to itself as well as all transitions (a, x, x'), such that there exist intermediate states y and y' with $R(a, y, y')$ and the states x, y and y, x' are connected by zero or more τ-transitions. In order to compute the representation of $\stackrel{\hat{\alpha}}{\Longrightarrow}$ we require the transitive, reflexive closure R_τ of R. A pair of states (x, x') is contained in R_τ, if the relation R contains an arbitrary sequence of τ-transitions connecting states x and x'. Thus R_τ can be computed as:

$$\begin{aligned} R'(x, x') &\equiv (x' = x) \vee R(\tau, x, x') \\ R_\tau &\equiv \mu Z [R'(x, x') \vee \exists\, y\, .\, [R'(x, y) \wedge Z(y, x')]] \end{aligned}$$

Using R_τ, the representation $\hat{R}$ of $\stackrel{\hat{\alpha}}{\Longrightarrow}$ is computed as:

$$\hat{R}(\alpha, x, x') \equiv (\alpha = \tau) \wedge (x' = x) \vee \exists\, y_1\, .\, \exists\, y_2\, .\, [R_\tau(x, y_1) \wedge (R(\alpha, y_1, y_2) \wedge R_\tau(y_2, x'))]$$

Given two transition systems $P(\alpha, x_1, x_1')$ and $Q(\alpha, x_2, x_2')$, the bisimulation-relation $S(x_1, x_2)$ is computed as the least fixpoint of:

$$\begin{aligned} S(x_1, x_2) \equiv \nu S\, [\forall \alpha, x_1'\, .\, [P(\alpha, x_1, x_1') \Rightarrow \exists\, x_2'\, .\, [\hat{Q}(\alpha, x_2, x_2') \wedge S(x_1', x_2')]] \\ \wedge\, \forall \alpha, x_2'\, .\, [Q(\alpha, x_2, x_2') \Rightarrow \exists\, x_1'\, .\, [\hat{P}(\alpha, x_1, x_1') \wedge S(x_1', x_2')]]] \end{aligned}$$

4 Automatic Translation of CCS Models to BDDs

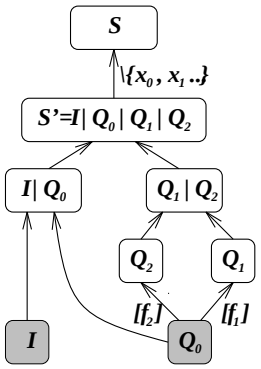
The goal of the translation process is to generate a BDD representation of the CCS-model in form of a transition relation $R(\alpha, x, x')$. With BDDs all actions α and states x, x' have to be encoded as bit patterns of fixed size. The encoding of actions is simple, as we can easily extract all occuring actions from the specification. Given a number k of different action symbols, $\lceil \mathrm{ld}(2k) \rceil$ binary variables are required to represent each action. The factor 2 stems from the fact that each action is associated with a co-action in the CCS-Model. Additionally we have to account for the τ-action which doesn't have a complement and is allways represented by the value $\alpha = 0$.

The decision on a suitable encoding of processes from the CCS-Model, which correspond to states of the transition system, is more difficult. First, we do not

know the total number of states in the final transition system. Thus it is not possible to determine an optimal encoding with respect to the number of binary variables. However, it is possible to determine the number of states in elementary processes. New processes are then constructed by means of parallel composition, renaming etc., and the state representations are derived from the state encodings of the processes involved in the operation. For example, in a parallel composition of two processes represented by transition relations $R_1(\alpha, x_1, x_1')$ and $R_2(\alpha, x_2, x_2')$, the states of the resulting system $R(\alpha, x, x')$ are simply given by concatenations $x = x_1|x_2$ and $x' = x_1'|x_2'$ respectively.

Thus, we are left with the problem of computing the number of states in simple processes. We first have to analyze the structure of the CCS specification to find all simple processes and sets of CCS equations that model their behaviour. Furthermore, not all of the states are given explicit process names in the CCS representation. Each '.' in a sequence of prefix operations, like $Q_0 = x_0.a_0.Q_0$ corresponds to a single, unnamed state. Therefore, the description of the system is first split up into a sequence of elementary CCS operations, as they appear in the definition of the grammar in 1, and each previously unnamed state is given an explicit name. Next, we determine the structure of the model, starting from the definition of the process, say S, that represents the whole system. We recursively collect the defining equations of all processes occurring in *sum-* or *prefix-* operations. Processes referenced in other operations, like composition or renaming etc., are stored as *dependencies*. In order for the specification to represent a finite state system, these dependencies among processes must imply a partial order. As a result of this analysis, we get a hierarchical structure of the specification. An example of such a structure is shown in Fig. 3. Processes I and Q_0 correspond to simple processes of this model.

The set of transitions of an elementary system can now be determined from its CCS description using the recursive functions shown in Table 1. *Def(E)* denotes the defining equation of a process E which is either of the form $E = a.E_1$ or $E = E_1 + E_2$. Function *Trans*$(\cdot)$ computes the set of all transitions for a given definition. To determine the set of transitions for a process which is defined as a sum, we have to rename the first component E_1 of all transitions (a, E_1, E_2) of the constituing processes. This renaming is computed by function $Link(\cdot, \cdot)$.



$$I \stackrel{\text{def}}{=} \bar{x}_0.\mathbf{0}$$
$$Q_0 \stackrel{\text{def}}{=} x_0.a_0.(\tau.\bar{x}_1.Q_0 + \bar{x}_1.\tau.Q_0)$$
$$Q_1 \stackrel{\text{def}}{=} Q_0[f_1]$$
$$Q_2 \stackrel{\text{def}}{=} Q_1[f_2]$$
$$S \stackrel{\text{def}}{=} (I \mid Q_0 \mid Q_1 \mid Q_2)\backslash\{x_0, x_1, \ldots\}$$

Fig. 3. Hierarchical structure of the translation of a CCS model

The notation $(a, E_1, E_2) :: l$ denotes a list of transitions consisting of a single transition $E_1 \xrightarrow{a} E_2$ followed by a transition list l.

$$\begin{aligned} Trans(E = a.E_1) &\equiv \{(a, E, E_1)\} \cup Trans(Def(E_1)) \\ Trans(E = E_1 + E_2) &\equiv Link(E,\ Trans(Def(E_1)) \cup Trans(Def(E_2))) \\ Link(E, (a, E_1, E_2) :: l) &\equiv \{(a, E, E_2)\} \cup Link(E, l) \\ Link(E, \emptyset) &\equiv \emptyset \end{aligned}$$

Table 1. Functions for the calculation of transition sets of elementary subsystems

Having computed the transitions of an elementary process, the number N of its states can be determined. Since these states are encoded as natural numbers in the range 0 to $(N-1)$, the vector of binary variables describing the process states has dimension $\lceil \mathrm{ld}(N) \rceil$. With this encoding of states and actions the BDD representing a process E is computed as the disjunction of its transitions:

$$R(\alpha, x, x') \equiv \bigvee_{(a,E_1,E_2) \in Trans(Def(E))} (\alpha = C(a)) \wedge (x = C(E_1)) \wedge (x' = C(E_2))$$

where function $C(\cdot)$ determines the encoding of its argument state or action.

5 Scheduler Specification in CCS

Suppose a set of $(N-1)$ processes P_i shall be controlled in such a way, that each process repeatedly executes an associated task with the condition that the processes are started sequentially. The CCS specification of a scheduler to control the execution of these tasks consists of an initialization process and N processes which communicate with each other via actions x_j [1]. To prove that the scheduler starts the processes P_i in the required order, a reference process must be defined which exhibits the specified, externally visible behaviour.

Table 2 contains computation times in seconds for different numbers N of composed processes. Column T_{COMP} contains the times required to compute the BDD representation of the composed model, T_{REACH} the time for computation of the set of reachable states and T_{BISIM} the computation times required to determine the set of bisimulating states. For comparison, the last column of the table contains computation results determined for the CWB [3] for models of size $2, 4$ and 8. All computations were carried out on a Sparc 10 with 32MB memory. As can be seen from these results, BDDs can easily handle systems with approximately $3 \cdot 10^6$ states and $3 \cdot 10^8$ transitions. Systems of this size are clearly out of reach of conventional state space investigation techniques.

N	states	transitions	T_{BISIM}	T_{COMP}	T_{REACH}	CWB
2	13	19	0,40	0,06	0,08	0,07
4	97	241	0,87	0,77	0,30	9,41
8	3073	13825	2,50	1,48	2,40	1615.17
12	73729	479233	7,20	1,93	7,27	–
16	1572865	13369345	16,32	2,80	18,63	–
20	31457281	330301441	28,50	3,77	39,28	–

Table 2. Verification results of the scheduler, times given in seconds

6 CSMA/CD protocol specification in CCS

The scheduler example indicates that it is possible to handle huge state spaces efficiently with BDDs. However, this example is mostly of academic interest. Therefore, we have modelled the CSMA/CD protocol, including propagation delay effects. The concurrency of the CSMA/CD protocol results from the common usage of a broadband transmission medium by several independently acting stations. Every station owns a MAC (Medium Access Controler) which controls the transmission of packetized data on the medium. The $MACs$ sense the medium for ongoing transmissions and may only start a new transmission if the medium is sensed idle. The protocol model gets more complicated if we take care of the finite packet propagation delay. We aim for a model as simple as possible, but we have to decide on the minimal number of stations we have to model in order to account for all relevant effects. We conclude that a model with three participating stations should be sufficient to capture all relevant effects and the specification should be complex enough for testing our BDD-tool.

The structure of the protocol model is depicted in Fig. 4. The transmission medium communicates with the $MACs$ which manage channel access inside the stations. To express the correct succession of all events, we have to model the events "packet-begin" and "packet-end passes a station". The following transitions have to be synchronized between a MAC and the medium:

b_i/e_i: MAC_i starts/ends the transmission of a packet

r_i/s_i: MAC_i receives the beginning/end of a packet

c_i: MAC_i detects a collision of transmitted packets on the medium

The following two events model communication between MAC and station:

$send_i$: MAC_i takes over a packet ready for transmission from its station

rec_i: MAC_i passes a packet received from the medium to its station

The complete protocol specification is composed of the medium M and the $MACs$. From a physical point of view the medium is absolutely passive, it accepts packets for transmission and passes them on. The ability to prevent and resolve collisions is assigned to the $MACs$. For every transmitted packet in a system with three stations there are at least six synchronisations of the medium with the $MACs$: three for the packet-header and three for the packet-tail. Collisions

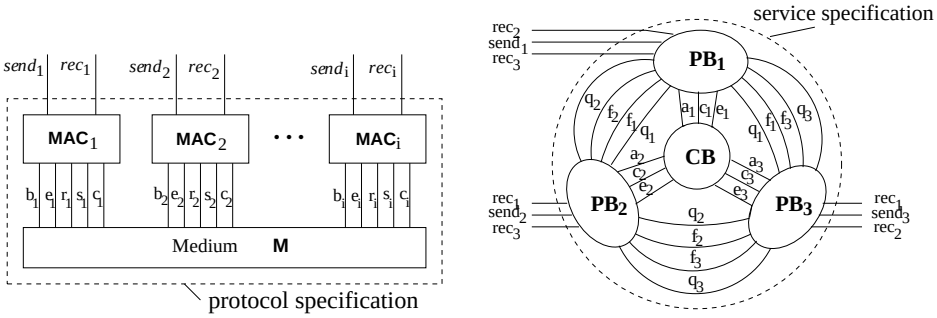


Fig. 4. Structure of service and protocol specification

generate any sum of additional collision signals. For a detailed description of the the equations and their semantics we refer to [9].

To prove the functional correctness of the protocol specification, we have to specify an appropriate reference process. This service specification should also model the propagation delay which results in a kind of storing of packets on the channel. The service specification is not in particular bound to any existing components and communication structure. We can design any component with any communication. We only have to secure the error-free packet transmission. To model the described virtual storage of packets on the channel and the MACs, the service specification consists of a two-layered buffer system where every station can deposit packets to be read by other stations. Every station can deposit one packet in the buffer system without restrictions, but at most one can enter a second packet. The complete buffer system consists of a separate pre-buffer *(PB)* for each station and a shared central buffer *(CB)* corresponding to the idealized model of the transport medium. Because of lack of space we do not give a full description of the service specification at this point. The structure of the service specification SS is also shown in Fig. 4. Note, that a direct communication between each component is possible (actions f_i,q_i). Of course, the observable transitions have to be the same as in the protocol specification: $send_1$, $send_2$, $send_3$, rec_1, rec_2 and rec_3.

6.1 Verification Results

Different from the previously described example of the scheduler, model and specification of the protocol have state spaces of almost equal size. Therefore, the following table of computation times contains separate numbers of states and transitions of both models.

process	states	transitions	T_{REACH}	T_{COMP}	T_{BISIM}	CWB
protocol	425	1122	1,93	7,52	447,8	52,9
service	484	1229	2,50			

As can be seen, times T_{COMP} required for the computation of the composed transition systems are again negligible. Considering the complexity of the protocol specification in comparison to the previously handled scheduler example, the number of states and transitions is surprisingly small. This results from a strong synchronization of MACs in the protocol model and PB-processes in the description of the service.

Computation of the bisimulation using BDDs is slower than the time required by the CWB by about a factor of 9. This is mainly caused by the complexity of the specification. While in the scheduler example the process describing the required behaviour contains only a few states, the protocol model and its specification have state spaces of almost identical size. This leads to large BDDs which are computed as intermediate results during the evaluation of the fixpoint equation for bisimulation and thus results in the observed computation times.

7 Conclusions

We have described the automatic translation of CCS models to a representation using BDDs and the application of BDD's to prove bisimulation equivalence for two example systems. In both cases the effort of generating a representation of the state space of the composed systems, which is often the primary bottleneck for the application of automatic verification methods, was negligible. Especially for the example of the scheduler model it could be shown that BDDs can handle state spaces which are far out of reach of conventional state space investigation techniques. Also the effort to determine bisimulation equivalence was small.

The effort to prove bisimulation equivalence in the case of the CSMA protocol, however, was much larger. This is not necessarily too surprising, because it is known from other work reported in the literature that in some cases explicit state space enumerating methods are more efficient than symbolic model checking using BDDs. The observation, that the proof of the scheduler is faster with BDDs than with the CWB, but slower for the CSMA-protocol should be further investigated. From this we expect some important ideas for the improvement of our BDD-tool. However, even with increased computation times a tool that is capable of handling huge states spaces is certainly preferable to a fast program that fails when applied to complex systems. Additionally, there are still some possibilities for improving efficiency which we have not yet investigated: for example splitting up the representation of the transition system into a set of smaller BDDs or a bisimulation preserving simplification of the model prior to bisimulation computation.

References

1. R. Milner: "Communication and Concurrency", Prentice Hall International Series in Computer Science, ISBN 0-13-114984-9.
2. R. Milner, J. Parrow, D. Walker: "A calculus of mobile processes", in Information and Computing 100, 1992, pp. 1-77.
3. R. Cleaveland, J. Parrow, B. Steffen: "The concurrency workbench: A semantics-based tool for the verification of concurrent systems", ACM Transactions on Programming Languages and Systems, 15(1):36-72, January 1993.
4. B. Krämer, G. Henze et al.: "Deriving ANSAware Applications from Formal Specifications", Proceedings of SDPS'95, 1995.
5. Randal E. Bryant: "Graph-based Algorithms for Boolean function manipulation", IEEE Trans. Computers, C-35(8): 677 - 691, August 1986.
6. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, J. Hwang: "Symbolic Model Checking: 10^{20} states and beyond". Technical Report, CMU, 1989.
7. K. L. McMillan: "Symbolic Model Checking: An approach to the State Explosion Problem.", PhD thesis, Carnegie Mellon Univeristy, 1992.
8. R. Enders, T. Filkorn, D. Taubner: "Generating BDDs for symbolic model checking in CCS", in Distributed Computing, 1993, 6:155-164.
9. K. Gotthardt, I. Scheler: "Formale Verifikation von Vielfach-Zugriffsprotokollen in CCS" GI-Fachtagung Softwaretechnik, Braunschweig, 1995

This article was processed using the LaTeX macro package with LLNCS style