

Formal Verification of Communication Protocols in Distributed Systems

Edgar Pek and Nikola Bogunović

Department of Electronics, Microelectronics,

Computer and Intelligent Systems

Faculty of Electrical Engineering and Computing

Unska 3, 10000 Zagreb

Phone: (01) 6129-999 int. 548 Fax: (01) 6129-653 Email: edgar.pek@fer.hr

Abstract—In distributed applications, software components embedded in the communication protocols collectively provide the interaction and functionality among various parts that run on the common pervasive platform. However, software components, as the target for the most of the changes, are expected to carry majority of the design faults. Hence, their verified conformance to the specification (correctness) is crucial for the reliable operation of the application. The paper surveys recent progress in the development of formal techniques as applied to the verification of protocols in distributed systems. Based on this analysis an appropriate method is selected and rigorously scrutinized by its practice to a specific protocol. Performance issues and possible improvements are discussed.

I. INTRODUCTION

Distributed systems provides the user with access to the various resources the system maintains. Access to shared resources allows computation speedup, and require more reliable and disperse communication. A distributed system must provide various mechanisms for interprocess communication, such as synchronization and deadlock prevention.

The majority of safety-critical systems are distributed, so it is necessary to ensure proper functioning of such systems. Formal methods provide mathematical procedures for describing and verifying system properties. Hence, formal verification is the process of ensuring that formal model of the design (*Imp* - implementation) satisfies a formal specification (*Spec*) with mathematical certainty, or plainly that an implementation conforms to specification. The possible conformance relations are equivalences, various simulation relations, (logical) satisfaction, (logical) implication, refinement, etc. Formal verification guarantees beyond any doubt the correct relation between the mathematical objects *Imp* and *Spec*. However, it does not establish the correctness of the real implementation abstracted by *Imp*.

Formally verifying computing algorithms and protocols are known to be a difficult task since interesting systems, modeled by a transition structure, will have millions of states. In the classical proof-based verification, the system description is a set of formulas Γ (in a suitable logic) and the specification is another formula φ . The verification method consists of trying to find a proof that $\Gamma \vdash \varphi$ (i.e. by applying allowable rules to the set Γ , derive φ). In a model-based approach [1], a finite model M in an appropriate logic represents the system. The specification is again represented by formula φ . The verification method consists of computing whether a model M logically satisfies φ (i.e. $M \models \varphi$). The model-based approach

is potentially simpler than the proof-based approach, for it is based on a single model M , rather than a possibly infinite class of them. In model checking one is not concerned with semantic entailment ($\Gamma \models \varphi$), or with proof theory ($\Gamma \vdash \varphi$), but rather with the notion of *satisfaction*, i.e. the satisfaction relation between the model and a formula ($M \models \varphi$).

This paper considers formal specification and verification of Bounded Retransmission Protocol (BRP) developed by Philips for the 4th generation infra-red control systems [2]. BRP represents an extension of the alternating bit protocol [3] for data packets transmission over lossy channel in limited amount of time.

The protocol has already been formally described and verified using various approaches. Groote and van der Pol used process algebra language μCRL (micro Common Representation Language) and appropriate proof theory [4] for protocol description and verification [5]. Proof was computer checked using Coq system.

I/O automata theory is a formalism based on extended finite state machines. This concept was used in [6] to specify and verify the protocol. Similarly, computer verification in type theory was performed using Coq .

Havelund and Shankar used combination of model checking and theorem proving for BRP verification. They obtained a down-scaled version and analyzed the protocol with *Murph*, tool based on state exploration. *Murph* specification was converted into PVS description. This description was generalized to full infinite state description and verified with PVS. Abstract description of the protocol was model checked using SMV [7].

In [8] BRP was specified using LOTOS (Language of Temporal Ordering Specification) and verified by CADP toolbox. Verification of the protocol was achieved by two techniques: bisimulations and temporal logic.

In [9] lossy channel systems model BRP, and the tool for checking safety properties is used for verification.

In [10] two tools are used to verify BRP, UPAAL and SPIN. The protocol is specified as network of timed automata, and in PROMELA as finite state machine. UPAAL is used to verify real-time aspects, and SPIN verifies protocol properties.

Dams and Gerth [2] describe the use of abstract interpretation techniques in model checking. They introduce canonical abstractions to obtain parametrized finite state system. Protocol abstractions and concrete model was implemented using PROMELA, and verified with SPIN.

Another combination of theorem proving and model check-

ing in BRP verification is presented [11]. In this paper inductive theorem prover INKA is used to confirm data abstractions. These abstractions are used when specifying an finite-state model appropriate for model checking.

In [12] VIS is used to verify protocol properties expressed in CTL (Computation Tree Logic). Timing aspects as well as structural aspects of the protocol were considered. Verilog were used as modeling language.

Our paper presents verification of the protocol and timing properties using NuSMV tool [13]. NuSMV is a symbolic model checker which represents an extension of SMV [7]. It has been chosen because it allows Real-Time CTL (RTCTL) model checking an quantitative temporal analysis [1]. NuSMV also implements integration of model checking techniques based on propositional satisfiability(SAT) [14]. We haven't used the SAT-based model checking in this work.

Qualitative and quantitative temporal analysis are presented in section II. In the following section we briefly describe implementation modeling formalism(NuSMV). Section IV gives the Bounded Retransmission Protocol service and protocol specification. Actual specification and verification procedures and results are described in the subsequent section. Finally, a conclusion gives some valuable comments on the conducted experiments and obtained results.

II. QUALITATIVE AND QUANTITATIVE TEMPORAL ANALYSIS

In this section we describe property specification languages CTL and its extension RTCTL, and show how to compute quantitative information about the model.

A formal specification language S provides a notation (syntactic domain), a universe of objects (semantic domain), and a precise rule defining which objects satisfy each specification. S is a triple $\langle Syn, Sem, Sat \rangle$, where Syn and Sem are sets and $Sat \subseteq Syn \times Sem$ is a relation between them. The more explicit specification language implies the better-defined formal method. Specification language that is fundamentally advantageous in specifying system behavior should belong to the family of temporal logic. In this paper we concentrate on the branching discrete time temporal logic, i.e. computation tree logic (CTL) formulae [15] and its extension real-time computation tree logic (RTCTL).

A. Qualitative temporal analysis

Formula in CTL temporal logic can be true for some states of the underpinning model (Kripke structure) and false for other states. The Kripke structure, or the model M , is defined as a triple $M = \langle S, R, L \rangle$, where S is a set of states, R is a transition relation in the structure, and L is a labeling function that assigns atomic propositional formulas to every state in S . Formulae may change their truth values as the system evolves from state to state. CTL formulae are built from atomic propositions, standard Boolean operators, and temporal operators. Atomic propositions express state properties of the system. Temporal operators describe when these properties have to be satisfied. Each temporal operator consists of a

path quantifier and a temporal modality. The path quantifier A indicates that a property is true for all computations (traces, paths), and E denotes that it is true for some computation (trace, path). The temporal modality describes ordering of events in time of computation, and can be one of the following: F (in some future and possibly current state), G (in every state), X (in the next state), U (strong until). In CTL one can easily express properties such as: "For any state, if a request (to deliver a message) occurs, it will eventually be granted":

$$AG[(request = true) \Rightarrow AF(acknowledged)]. \quad (1)$$

Any CTL formula φ could be interpreted within the given Kripke structure M as denoting a set of states, namely those states Q for which $(M, s \models \varphi)$ holds, or formally:

$$Q(\varphi) \subseteq S \text{ is a set of states: } Q(\varphi) = \{s \mid M, s \models \varphi\}. \quad (2)$$

Usually we require that the initial state is included, i.e. $s_0 \in Q(\varphi)$. Clearly, the verification procedure encompasses exploration of states as the system evolves from the initial state s_0 in discrete time steps.

The kind of practically relevant properties that need to be checked fall in to two main categories. *Safety* (invariant) properties ensure that nothing bad ever happens in the considered system. A failure can be demonstrated by a finite transition sequence. E.g.: "From any state it is possible to get to the state where restart=True":

$$AG(EF restart = true). \quad (3)$$

Liveness (progress) properties ensure that something good should eventually happen. A failure can be demonstrated only by an infinite transition sequence. An example of liveness property is given by the Eq. 1.

B. Quantitative temporal analysis

If we want to express complex timing properties of the system, CTL is not suitable. Because, in CTL we cannot easily express that "every request req should be met with a response ack within n time" units. Simple and effective way to perform verification of time bounded properties is to introduce bounds in CTL temporal operators [1]. The extended logic is called RTCTL (Real-time CTL). Notice, that RTCTL is expressive as CTL because time bounded operations can be translated into nested EX (AX) operators. This translation is often impractical and error-prone.

The basic RTCTL operator is the *bounded until* operator which has the form: $U_{[a,b]}$ where $[a,b]$ defines the time interval in which the property must be true. Formula $fU_{[a,b]}g$ is true of some path $\pi = s_0, s_1, \dots$ if g holds in some future state s on the path, f is true in all states between s_0 and s , and the distance from s_0 to s is within the interval $[a, b]$.

Using RTCTL we can easily express properties such as: "Every request from a client should be met with a response from the server within five time units". This property can be expressed in RTCL as $AG(req \rightarrow AF_{[0,5]}resp)$.

Along with the verification of time bounded properties when timing constraints are known we are interested in computing

quantitative timing information such as exact minimum and maximum delays between transitions in discrete model. The algorithms are designed to work well with symbolic techniques based on the use of binary decision diagrams.

Both, a minimum delay and a maximum delay algorithm take as input a Kripke structure $M = (S, R, L)$ and two set of states *start* and *final*. The minimum delay algorithm returns the length of shortest path from a state in *start* to a state in *final*. In no such path exists the algorithm returns *infinity*. The maximum delay algorithm returns the length of a longest path from a state *start* to a state *final*. If there exists an infinite path beginning in a state *start* that never reaches a state *final*, the algorithm returns *infinity*.

III. NuSMV

NuSMV is a symbolic model checker originated from reengineering, reimplementing and extension of CMU *SMV*, the original BDD-based model checker developed at CMU. NuSMV allows for the representation of specifications expressed in CTL and Linear Temporal logic (LTL) using BDD-based and SAT-based model checking techniques. Also we have used NuSMV capability to verify RTCTL formulas and to compute quantitative information on the FSM.

The primary purpose of the NuSMV input language is to describe the transition relation of a finite Kripke structure. The input file describes both the model and the specification (with possible fairness constraints). The states are defined by a collection of state variables, which may be of Boolean or scalar type. The transition relation of the Kripke structure is determined by a collection of parallel assignments, which are introduced by a keyword *ASSIGN*. The semantics of assignment in NuSMV is similar to that of a single assignment data flow languages. A program can be viewed as a system of simultaneous equations, whose solutions determine the next state. When a set is assigned to a variable, the result is a non-deterministic choice among elements of the set. Non-deterministic choices are useful for describing systems which are not yet fully implemented, or abstract models of complex protocols, where the value of some state variables cannot be completely determined.

IV. THE BOUNDED RETRANSMISSION PROTOCOL

In this section we provide the service specification (Fig. 1) and the protocol specification. The service specification describes how the protocol and an environment are related. The protocol specification (Fig. 2) describe communication between the protocol entities and behavior of each protocol entity.

A. Service specification

In the BRP service specification we consider four entities: *Producer*, *Sender*, *Receiver* and *Consumer*. In Fig. 1 and Fig. 2 it can be seen how these entities communicate. In essence the producer wants to deliver large data packet through lossy channel to the consumer. Data packet is delivered as sequence of chunks. For each chunk there is a limited amount

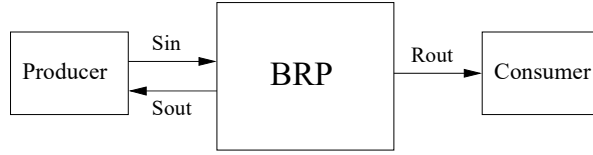


Fig. 1. The file transfer service specification

of time available for delivery to the consumer. Protocol cannot guarantee successful delivery within given time bound. The consumer receives either whole data packet or the first k chunks. The producer and the consumer are informed if whole data packet was delivered successfully or not.

The producer supplies data packet through S_{in} (Fig. 1), for each packet it receives an indication through S_{out} . If data packet was delivered successfully to the consumer it receives I_{OK} . In case of the unsuccessful delivery it receives I_{NOK} . When the sender cannot be sure if the last chunk was delivered to the consumer, the producer receives the indication I_{DK} .

For each data packet, and for each chunk the receiver provides an indication to the consumer through R_{out} . The consumer receives I_{FST} if a chunk is the first, and there is at least one more chunk in the data packet. The indication I_{INC} means that chunk is intermediate. The indication I_{OK} informs the consumer that chunk is last element of the packet. If delivery has been aborted due to loss of a chunk receiver supply the I_{NOK} indication.

B. Protocol specification

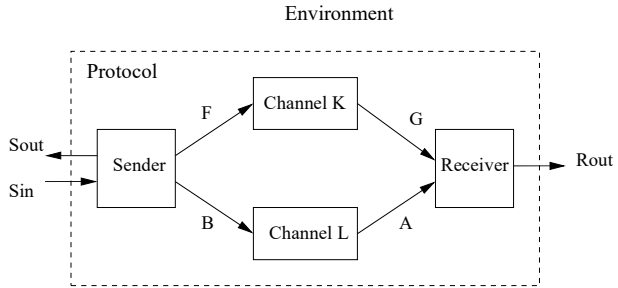


Fig. 2. The BRP specification

In our specification the BRP can be represented with four entities: *Sender*, *Receiver*, *Channel_K* and *Channel_L*. The relations between entities are shown in Fig. 2. There are also three timers. The sender is connected with timers $Timer_X$ and $Timer_{SYNC}$, and the receiver with timer $Timer_Z$. The environment and the protocol correspond through S_{in} , S_{out} and R_{out} as described in the subsection IV-A. The sender transmit every chunk of the data packet through lossy channel (*Channel_K*) to the receiver. The receiver acknowledge each received chunk through lossy channel (*Channel_L*).

The sender transmit one chunk at a time. Each chunk consists of an indication and an alternating bit along with datum. Indication can be: I_{FST} , I_{INC} and I_{OK} . Indication I_{FST} represents first chunk of data packet which consist of at

least two chunks. I_{INC} informs that the chunk is neither the first nor the last in the packet. I_{OK} means that the chunk is the last one in the data packet. Alternating bit enables that the receiver can detect duplicate chunks of the same data packet. Duplicate chunks have same the alternating bit, and such chunks are not delivered to the environment (consumer), but nevertheless acknowledged. The sender starts $Timer_X$ right after it sends the chunk to the receiver. It waits for an acknowledgement from the receiver or a timeout of $Timer_X$. In case of the timeout it retransmits the chunk. However, if number of unsuccessful retransmissions reach a bound, the sender aborts transfer of data packet currently being sent. After the abortion it starts $Timer_{SYNC}$ and waits for its timeout before the next delivery of an data packet. This is needed because the receiver must properly react to abortion.

The receiver starts $Timer_Z$ when it receives a new chunk from the sender. It acknowledges the chunk and waits for new one. If $Timer_Z$ times out, it means that new chunk has not been received for a long time. So, the receiver can be sure that the sender has cancel the delivery.

V. FORMAL PROTOCOL SPECIFICATION AND VERIFICATION IN NUSMV

In this section we first describe the formal BRP model implemented in NuSMV. As an example of NuSMV specification we provide an implementation of the *Producer* module in Fig. 3. In the second subsection we present an CTL verification of service and protocol properties and quantitative analysis of the BRP protocol.

A. BRP model

The BRP model implementation is obtained using SDL description provided in [12]. Our NuSMV model consists of six synchronous modules: *Producer*, *Sender*, *Channel*, *Receiver*, *Consumer* and *Timer*. There are two instances of the *Channel* module: *Channel_K* and *Channel_L*. Also, we have three instances of the *Timer* module: *Timer_X*, *Timer_Z* and *Timer_SYNC*.

The *Producer* module (Fig. 3) consists of three state variables n , $state$ and req . The variable n stores randomly chosen number of chunks in the data packet. Using req , the *Sender* module is informed that delivery of the data packet starts. Variable req , as expected holds a state of the *Producer* module. After the $ready$ signal is received from *Sender*, *Producer* goes to state $wait_conf$, where it waits for information ($conf$) from the *Sender* about delivery of the data packet.

Similarly, we have obtained the NuSMV description for other five modules presented in our formal specification of the BRP.

B. Verification and quantitative analysis

Qualitative analysis of the protocol has been performed using CTL specifications. Basic safety and liveness consideration are related with the service specification for which we have verified many CTL formulas. Some of them are listed in Fig. 4

```
MODULE Producer(conf, ready)
VAR
  n : {1, 2, 3, 4};
  state : {start, wait_conf};
  req : boolean;
ASSIGN
  init(n) := {1, 2, 3, 4};
  init(state) := start;
  init(req) := 0;
  next(state) :=
    case
      state = start & ready=1 : wait_conf;
      state = start & ready=0 : start;
      state = wait_conf &
conf=NONE : wait_conf;
      state = wait_conf & conf!=NONE : start;
    esac;
  next(req) :=
    case
      next(state) = wait_conf : 1;
      1 : 0;
    esac;
  next(n) :=
    case
      state=start : {1, 2, 3, 4};
      1 : n;
    esac;
```

Fig. 3. NuSMV code for the producer

```
# 1. Liveness property
SPEC AG AF (Sender_pr.ready=1->
AX(Producer_pr.req=1))
# 2. First indication
SPEC
AG (Receiver_pr.state=new_file->
AX A[Rout=NONE U
(Rout=I_FST|Rout=I_OK|Sender_pr.ready=1)])
# 3. First indication cannot be I_NOK
SPEC
AG (Receiver_pr.state=first->
AX A[Rout=NONE U Rout!=I_NOK])
# 4. Last indication
SPEC AG((Rout=I_OK | Rout=I_NOK)->
AX A[Rout=NONE U Sender_pr.ready=1])
```

Fig. 4. CTL formulas for service specification

The first CTL formula represents liveness property. It states that the producer produces data packet and the sender start delivery of them infinitely often. The second formula describes that first received chunk can be either I_{FST} or I_{OK} . The third formula states that the first received chunk cannot be I_{NOK} . When I_{OK} or I_{NOK} is received we can be sure that this is the last indication.

In the sequel we perform discrete quantitative analysis to verify protocol specification. The BRP protocol critically depends on timeout intervals, so it is important to formally specify timeout intervals.

We can define $Time_X$, $Time_Z$ and $Time_{SYNC}$ as timeout intervals for *Timer_X*, *Timer_Z* and *Timer_SYNC* respectively.

$Time_X$ has to be greater than the maximum time needed for a chunk sent by the sender to get to the receiver and for acknowledgement to come back if neither the chunk nor acknowledgments are lost. The computation of the $Time_X$ is shown in Fig. 5. The first formula represents time delay

```
# 1.
COMPUTE
MIN[Sender_pr.send_CK=1 & ChannelK.output=0,
ChannelK.output=1]
# 2.
COMPUTE
MIN[Receiver_pr.state=new_file &
Receiver_pr.send_CL=0,
Receiver_pr.send_CL=1]
# 3.
COMPUTE
MIN[Receiver_pr.send_CL=1 & ChannelL.output=0,
ChannelL.output=1]
# 4.
COMPUTE
MIN[Sender_pr.state=wait_ack,
Sender_pr.state=success]

# 5.
SPEC
AG (StartTimer_X=1 ->
!EX E[ FrameLost=0 U
(FrameLost=0 & Timeout_X=1)])

-- the result of
MIN [(Sender_pr.send_CK=1&
ChannelK.output=0),
ChannelK.output = 1 ]
is 2

-- the result of
MIN [(Receiver_pr.state = new_file &
Receiver_pr.send_CL=0),
Receiver_pr.send_CL=1]
is 3

-- the result of
MIN [(Receiver_pr.send_CL = 1 &
ChannelL.output = 0),
ChannelL.output = 1 ]
is 2

-- the result of
MIN [ Sender_pr.state = wait_ack,
Sender_pr.state = success ]
is 1
```

Fig. 5. Computation of the timeout interval of $Timer_X$

in the $Channel_K$, the second defines how long is needed for the receiver to process received message. The third formula represents time delay in $Channel_L$. Finally, the fourth formula represents time needed for the sender to receive an acknowledgment. For our model $Time_X$ has to be at least 9. CTL formula 5 states that timeout of $Timer_X$ means that chunk has been lost.

Timeout for the $Timer_Z$ appears if new chunk has not arrived for a long time, that indicates that the sender canceled transmission of the data packet. Computation of the timeout interval for the $Timer_Z$ is more difficult with respect to the $Timer_X$. It depends on the number of allowed retransmissions for the chunks and on the expected time interval between the end of delivery of one and start of delivery of the next data packet.

```
-- 1.
COMPUTE
MIN[StartTimer_Z=1,
Sender_pr.rc=2 &
Sender_pr.state=next_frame];

-- 2.
COMPUTE
MIN[Sender_pr.rc=2 &
Sender_pr.state=next_frame,
Sender_pr.rc=1 &
Sender_pr.state=next_frame];

-- 3.
COMPUTE
MIN[Sender_pr.rc=1 &
Sender_pr.state=next_frame,
Sender_pr.rc=2 &
Sender_pr.state=next_frame];

-- 4.
COMPUTE
MIN[Sender_pr.rc=2 &
Sender_pr.state=next_frame,
ChannelL.state=transfer];
-- 5.
SPEC AG ( StartTimer_Z ->
!EX E[PacketLost=0 U (PacketLost=0 &
Timeout_Z=1)] )

COMPUTE
MIN [StartTimer_Z=1, StartTimer_SYNC=1]

SPEC
AG (StartTimer_SYNC ->
AX A[ Timeout_SYNC=0 U
Receiver_pr.state=new_file] )

-- the result of
MIN [StartTimer_Z=1, StartTimer_SYNC=1]
is 5
```

Fig. 6. Computation of the timeout interval of $Timer_Z$

Fig. 7. Computation of the timeout interval of $Timer_{SYNC}$

In Fig. 6 we can see how timeout interval is computed. Maximum time interval in which the loss of the packet occurs starts when acknowledgement for the $MAX_RETR - 1$ retransmission is lost in the $Channel_L$. In that case, first timeout for $Timer_X$ expires and the sender starts the final retransmission of the chunk. Worst case scenario occurs when

following transmission is successful. Note, that $Timer_Z$ is not stopped because the receiver has already received this chunk. During following $MAX_RETR - 1$ attempts timeout can not occur. It can occur after the last allowed retransmission which is lost in $Channel_K$. We have computed that $Time_Z$ is 34 for $MAX_RETR = 2$. For $MAX_RETR = 3$, $Time_Z$ is 58.

Computation of the $Time_{SYNC}$ interval depends only on the interval $Time_Z$, and it can be stated as $Time_{SYNC} = Time_Z - 4$ for our model, because it takes at least five state transitions between start of $Timer_Z$ and $Timer_{SYNC}$. Computation of the timeout interval for the $Timer_{SYNC}$ is shown in Fig. 7.

VI. CONCLUSION

The main novelty of this paper is that it presents the discrete quantitative analysis of the protocol. Exact computation of timeout intervals is provided. Complex timing behavior of the protocol significantly complicates the verification. This analysis is appropriate method in synchronous protocol verification.

Bounded retransmission protocol verification provides useful insights for the further analysis of real-time distributed systems, which are often used in safety-critical system.

Future work will consider asynchronous protocol specification, and verification based on tools that can model continuous real-time.

REFERENCES

- [1] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. Cambridge, Massachusetts: The MIT Press, 1999.
- [2] D. Dams and R. Gerth, "The bounded retransmission protocol revisited," in *Electronic Notes in Theoretical Computer Science*, F. Moller, Ed., vol. 9. Elsevier Science Publishers, 2000.
- [3] A. S. Tanenbaum, *Computer Networks*. Prentice-Hall International, Inc., 1996, ISBN: 0-13-394248-1.
- [4] J. Groote and A. Ponse, "Proof theory for μCRL : a language for processes with data," in *Proceedings of the International Workshop on Semantics of Specification Languages*, ser. Workshops in Computing Series, Andrews et al., Eds., 1994, pp. 231–250.
- [5] J. Groote and J. v. d. Pol, "A bounded retransmission protocol for large data packets. A case study in computer checked verification," Tech. Rep. 100, 1993, <ftp://ftp.phil.uu.nl/pub/logic/PREPRINTS/preprint100.ps.Z>, Also appeared as [18].
- [6] L. Helmink, M. Sellink, and F. Vaandrager, "Proof-checking a data link protocol," in *Proceedings International Workshop TYPES'93*, Nijmegen, The Netherlands, May 1993, H. Barendregt and T. Nipkow, Eds., vol. 806, 1994, pp. 127–165.
- [7] K. McMillan, *The SMV system*, 1992 (2001). [Online]. Available: <http://http://www-2.cs.cmu.edu/~modelcheck/smv.html>
- [8] R. Mateescu, "Formal description and analysis of a bounded retransmission protocol," in *Proc. of COST 247 International Workshop on Applied Formal Methods in System Design*, Maribor, Slovenia, ser. Technical Report, Z. Brezocnik and T. Kapus, Eds. University of Maribor, 1996, pp. 98–113.
- [9] P. Abdulla and M. Kindhal, "Using lossy channel systems for the verification of communication protocols," in *Proc. of COST 247 International Workshop on Applied Formal Methods in System Design*, Maribor, Slovenia, ser. Technical Report, Z. Brezocnik and T. Kapus, Eds. University of Maribor, 1996, pp. 128–134.
- [10] P. D'Argenio, J.-P. Katoen, T. Ruys, and J. Tretmans, "Modeling and verifying a bounded retransmission protocol," in *Proc. of COST 247 International Workshop on Applied Formal Methods in System Design*, Maribor, Slovenia, ser. Technical Report, Z. Brezocnik and T. Kapus, Eds. University of Maribor, 1996, pp. 114–127.
- [11] D. Dams, D. Hutter, and N. Sidorova, "Using the INKA prover to automate safety proofs in abstract interpretation - A case study," in *Workshop on Modelling and Verification, C.I.S., Besancon, France, 1999.*, F. Bellegarde and O. Kouchnarenko, Eds.
- [12] R. Meolic, T. Kapus, and Z. Brezocnik, "Exploring properties of a bounded retransmission protocol with VIS," *CIT. Journal of Computing and Information Technology*, vol. 7, no. 4, pp. 311–321, 1999.
- [13] R. Cavada, A. Cimatti, E. Olivetti, M. Pistore, and M. Roveri, *NuSMV 2.1 User Manual*, 2002. [Online]. Available: <http://nusmv.first.itc.it/NuSMV/userman/v21/nusmv.pdf>
- [14] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *Tools and Algorithms for the Construction and Analysis of Systems. Part of European Conferences on Theory and Practice of Software, ETAPS'99, Amsterdam*, ser. LNCS, W. R. Cleaveland, Ed., vol. 1579. Springer-Verlag, 1999, pp. 193–207.
- [15] E. A. Emerson, "Temporal and modal logic," in *Handbook of Theoretical Computer Science*, J. van Leeuwen, Ed. New York, N.Y.: Elsevier Science Publishers B.V.: Amsterdam, The Netherlands, 1990, vol. B: Formal Models and Semantics, ch. 14, pp. 996–1072.
- [16] M. R. A. Huth and M. D. Ryan, *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge, England: Cambridge University Press, 2000.
- [17] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "NuSMV 2: An OpenSource tool for symbolic model checking," *Lecture Notes in Computer Science*, vol. 2404, pp. 359–362, 2002.
- [18] J. Groote and J. v. d. Pol, "A bounded retransmission protocol for large data packets. A case study in computer checked verification," in *Proceedings of AMAST'96*, M. Wirsing and M. Nivat, Eds., vol. 1101, 1996, pp. 536–550, also appeared as [5].