

Modeling and Verification of a Distributed Transmission Protocol

Lubomir Ivanov

Department of Computer Science, Iona College, 715 North Avenue, New Rochelle, NY 10801

livanov@iona.edu

ABSTRACT

Series-parallel poset verification is a powerful methodology for proving the design correctness of complex systems and protocols. In this paper we use series-parallel posets to model and verify the behavior of a distributed transmission protocol with multiple senders and receivers. The verification is carried out with the SPPV environment.

Keywords: *formal verification, series-parallel poset, distributed transmission protocol*

1. INTRODUCTION

The ever-increasing complexity of modern hardware and embedded systems has spurred the development of formal verification as an alternative/complement to classical simulation and testing methods. Over the past fifteen years, formal verification has established itself as a standard phase in the industrial design and development of complex hardware and software systems. The search for more efficient verification methodologies has evolved in two directions: powerful, general techniques capable of accurately modeling and verifying any system at the expense of the relatively high computation complexity of the underlying algorithms [1-3], and simpler, less expressive methodologies, applicable only in certain cases, but endowed with superior time/space complexity [4-7]. In [8-18] we introduced a new, efficient formal verification method – Series-Parallel Poset Verification – and demonstrated its usefulness in verifying the behavior of complex real-world systems and protocols. The verification is carried out with the SPPV Verification Environment – a software-based on the series-parallel poset methodology. The SPPV Environment provides a convenient GUI interface (accessible as a stand-alone application or as an applet on the web), which allows a user to specify a system behavior and a set of properties to be verified, and initiate the automatic verification process. If a particular property is not satisfied, an explanation is provided to help the designer with the debugging process.

In this paper we present a series-parallel poset model of a distributed transmission protocol based on the

Philips audio control protocol described in [20, 21]. The protocol synchronizes the transmission of messages from multiple senders to one or more receivers. We demonstrate the formulation of the protocol behavior expression, consider two verification properties, and discuss the overall verification of the protocol using the SPPV environment.

The paper is organized as follows: We begin by reviewing the most essential aspects of series-parallel poset verification and the SPPV Verification Environment. We then present the series-parallel poset model of the distributed transmission protocol. Next, we study the verification of the protocol, and compare our verification results to those of similar verification studies. The last section summarizes the presented material, and outlines directions for future work.

2. SERIES PARALLEL POSET VERIFICATION

2.1. Methodology

Series-parallel poset verification is based on the notion of a *partially ordered set (poset)*, which is a set with a reflexive, antisymmetric, and transitive relation defined on the set elements. With the help of two operations – concatenation (\bullet) and shuffle (\otimes) – a *series-parallel poset* over an alphabet Σ is defined inductively as follows:

- The empty poset, I , is a series-parallel poset
- For each $\sigma \in \Sigma$, the singleton poset labeled σ is a series-parallel poset
- If P and Q are series-parallel posets, so are $P \bullet Q$ and $P \otimes Q$

The set of all series parallel posets formed from I and the singletons, and closed under concatenation and shuffle forms a bimonoid denoted $SP(\Sigma^*)$ [19]. For purpose of formal verification, the alphabet, Σ , consists of all distinct events occurring in the system whose behavior is to be verified. The concatenation operation is used to indicate event sequencing, whereas shuffle denotes event independence. If e_1 and e_2 are two events, then their concatenation $e_1 \bullet e_2$ represents the fact that event e_1 occurs before event e_2 , while the shuffle $e_1 \otimes e_2$ indicates that both events occur but in

an unspecified order.¹ The sequencing and independence of events extends to sets of events:

Two sets of events, P and Q , are *independent* if no event in P triggers a chain of events leading to the occurrence of an event in Q and v.v., i.e. P and Q are independent if the set of predecessors of P does not involve any event from Q and vice versa. A set of events P *always precedes* a set of events Q if all events in P occur before any event in Q does, i.e. if each event in Q has all events in P as predecessors. A set of events P *partially precedes* a set of events Q if P sometimes occurs before Q . This is so when each event in Q has at least one predecessor from P , or when P and Q are independent.

Let us illustrate the definitions with an example. Consider the following series-parallel poset:

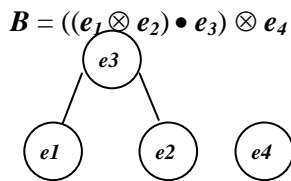
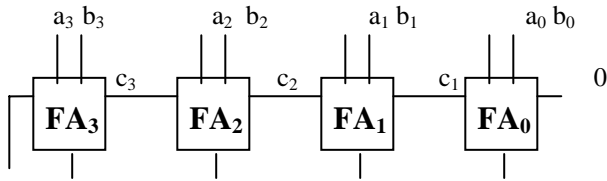


Fig.1 A Simple Series Parallel Poset Example

Consider now the sets of events $P_1 = \{e_1, e_4\}$, and $P_2 = \{e_3\}$. Clearly, P_1 and P_2 are not independent since e_1 must occur before e_3 does. P_1 does not always precede P_2 since one possible sequence of events is e_1, e_2, e_3, e_4 . But P_1 may sometimes precede P_2 since another possible sequence is, for example, e_1, e_2, e_4, e_3 .

Interpreting series-parallel posets as descriptions of the dependence or independence of sets of events allows us to model the *behavior* of a system in terms of the sequences of events occurring during its operation. In [8] we presented an approach to modeling the behavior and properties of non-iterated systems with series-parallel posets. A *non-iterated system* is one, in which events are distinct and not repeated.²



¹ For a formal definition of shuffle, concatenations, and other concepts introduced below, refer to [8, 10, 13].

² Not all non-iterated systems can be expressed with series-parallel posets. See [8] for details.

c₄ s₃ s₂ s₁ s₀
Fig.2 A Non-Iterated System: 4-bit Binary Adder

Fig.2 presents an example of a non-iterated system – a 4-bit binary adder. If we label the change of the sum and carry outputs due to a change in the inputs as events, the behavior of the above non-iterated system can be modeled as the following series-parallel poset expression:

$$B = (e_1 \otimes (e_2(e_3 \otimes (e_4(e_5 \otimes (e_6(e_7 \otimes e_8))))))))$$

For a more detailed explanation of this and other examples, refer to [8].

An *iterated system* is one in which some or all events are repeated. Thus, an iterated system consists of a number of components, which function in series or independently so that each component is either an iterated- or a non-iterated system. A wide variety of systems can be considered iterated: communication-, interconnect, and cache protocols, sequential circuits, feedback control systems, etc.

To describe behavior of *iterated systems*, a generalization of the idea of series-parallel posets is needed:

- The shuffle and concatenation operations are redefined to operate over sets of posets
- Two new operations are introduced – plus (+) and star (*)

In the context of verification, + is used to denote that either one or another event sequence occurs, i.e. it represents a choice of event sequences. The star operation (modeled after the Kleene star) is used to represent repetition of event sequences, i.e. the possibility that a sequence of events occurs 0 or more times. Thus, the set of finite subsets of $SP(\Sigma^*)$ closed under the above four operations defines the star-shuffle semiring $\mathcal{S} = (\mathcal{S}, +, \bullet, \otimes, *, \theta, I)$.

Formal verification requires that three components be specified:

- *System Behavior*: A mathematical model of the system
- *Verification Properties*: A set of expressions – one for each of the properties to be tested.
- *Proof Method*: A formal technique to allow each property to be verified within the system behavior

In the context of series-parallel poset verification of iterated systems, the first two of the above “ingredients” are defined as follows:

- The *Behavior*, B , of an iterated system is an element of the star-shuffle semiring S . Thus, the behavior of an iterated system is a set of series-parallel posets.
- A *Verification Property* is also a set of series-parallel posets but over a subset of the alphabet Σ .

There are four normal forms of behavior and property expressions:

- Concatenation: $B = B_1 \bullet \dots \bullet B_n$ & $P = P_1 \bullet \dots \bullet P_m$
- Shuffle: $B = B_1 \otimes \dots \otimes B_n$ & $P = P_1 \otimes \dots \otimes P_m$
- Plus: $B = B_1 + \dots + B_n$ & $P = P_1 + \dots + P_m$
- Star: $B = B_1^*$ & $P = P_1^*$

Before the verification process begins, the behavior expression is reduced to include only the events of interest specified in the property. This simplifies the reasoning about sets of events and the complexity of the verification algorithms [10, 13].

The third “ingredient” – the proof method – is implicit in the *verification questions* that can be asked about a property. The verification questions are specified as *predicates*:

- $SS(B, P)$ is a binary predicate, interpreted as “The property P is sometimes satisfied within the behavior, B ”. The predicate takes a behavior and a property and verifies that the property can sometimes be traced within the behavior.
- $AS(B, P)$ is a binary predicate, interpreted as “The property P is always satisfied within the behavior, B ”. The predicate takes a behavior and a property and verifies that the property can always be traced within the behavior.

The formal definitions of the two verification predicates $SS(B, P)$ and $AS(B, P)$ are quite complex, and examine the conditions for satisfiability of all four types of property normal forms with respect to the four types of behavior normal forms. Refer to [13] for the formal definition, correctness, and complexity arguments of the predicates and the verification algorithms that they give rise to.

2.2. The SPPV Formal Verification Environment

The SPPV Formal Verification Environment is an integrated software tool, based on the series-parallel poset verification methodology outline above. SPPV is

Java based, and can, therefore, be easily ported to any system. It is available as a stand-alone application, or as web-based applet. The SPPV Environment can be used for verifying properties pertaining to the correct sequencing of events in the broadest possible sense in a variety of iterated and non-iterated system. We have used the software for the verification of communication and cache coherence protocols, control microcode, integer pipeline operation. The advantages of using SPPV over other commercial verification products are its extremely efficient operation³, fully automated verification with no need for user interaction, and a simple, easy to use graphics user interface (Fig.3).

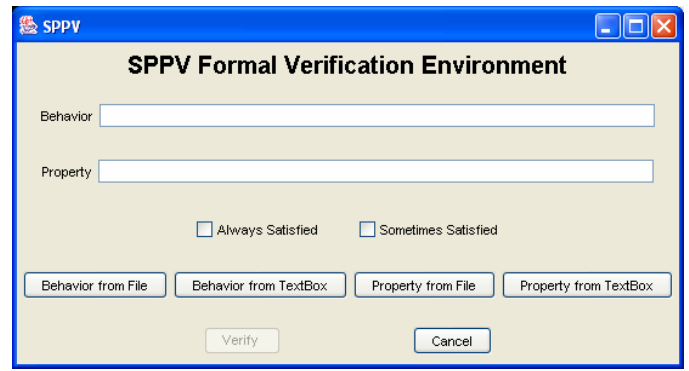


Fig.3 The SPPV Environment

3. MODELING OF THE DISTRIBUTED TRANSMISSION PROTOCOL

A protocol is a set of rules, which communicating devices have to follow in order to exchange information in a meaningful and unambiguous way. To ensure this, the design of protocols must be carefully verified. Protocol verification involves checking that certain events (e.g. issuing of commands and signals) occur in proper order, while other events are mutually independent. Our series-parallel poset modeling and verification methodology is well suited for this task, and, in what follows, we demonstrate how it can be used to model and verify the behavior of a distributed transmission protocol.

The distributed transmission protocol we chose to model and verify is based on the Philips audio control protocol with bus collision, used to send control

³ $O(n+m^3)$ worst case, $O(n+m^2)$ average case time complexity, and $O(m)$ space complexity, where n is the number of events in the behavior, and m is the number of events in the property being verified, which is usually very small

messages to various audio components [20, 21]. The protocol synchronizes the transmission of data from multiple sources over a single data line with one (or more) receivers. Each sender broadcasts its message over the data line using Manchester encoding. The bits are transmitted halfway through the bit slot, with an upward transition used for transmitting a ‘1’ and a downward transition for a ‘0’. If the same bits have to be transmitted one after the other, the voltage is changed at the end of the first bit slot. The message is preceded by the sender’s unique address, which must begin with a ‘1’. It is imperative, for bus collision resolution purposes, that no address be a prefix of another address. Thus, if one of the senders begins transmitting its address as “11...” and the other begins sending “10...”, on the second bit slot, a collision will be detected by the first sender when it senses that the voltage on the data line is high even though the first sender has transitioned its output to low (in preparation for transmitting the second ‘1’). Thus the first sender will terminate its transmission, and wait for a time before re-transmitting.

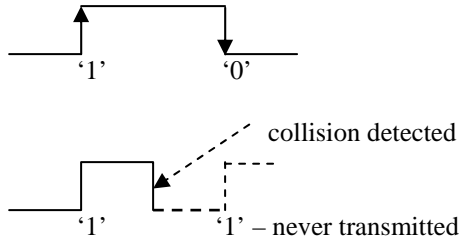


Fig.4 Collision Detection During Transmission

The behavior of a sender described above is given by the following series-parallel poset expression:

$$B_{TX_i} = (clk_0 \cdot (idle_{TX_i} + (in_{i-1} \cdot (sns1_{iL} \cdot clk_{1/2} \cdot d_H + sns1_{iH} \cdot d_L \cdot (sns2_{iL} \cdot clk_{1/2} \cdot d_H + sns2_{iH} \cdot rtx_i))) + in_{i-0} \cdot (sns1_{iH} + sns1_{iL} \cdot d_H) \cdot clk_{1/2} \cdot d_L))^*$$

If a ‘1’ is to be transmitted, then during the first half of the bit-slot (clk_0 to $clk_{1/2}$) the sender senses whether the data line is high or low. If the data line is low, then during the second half of the bit-slot ($clk_{1/2}$ to clk_0) the sender sets the data line to high. If the data line is sensed to be high, then the sender sets it to low to prepare for a 0-to-1 transition for sending the second ‘1’. At this point, the sender must test the voltage on the data line one more time: if, after setting it to low, the data line voltage is still high, there is another sender transmitting simultaneously. In that case, a retransmit signal is generated, and the transmission is terminated. Otherwise, if the voltage is determined to

be low, during the second half of the bit-slot the data line is raised high, transmitting a ‘1’ during the transition.

If a ‘0’ is to be transmitted and the data line is already high, during the second half of the bit-slot, the data line is set low. If the data line is sensed to be low, then it is raised high, followed by a reset back to low during the second half of the bit-slot. If no messages are to be transmitted, the sender is idle.

The series-parallel poset expression describing the behavior of the receiver is somewhat simpler:

$$B_{RX_j} = (clk_0 \cdot clk_{1/2} \cdot (sns1_L \cdot sns2_H \cdot out_{i-1} + sns1_H \cdot sns2_L \cdot out_{i-0} + sns1_L \cdot sns2_L \cdot idle_{RX_j}))^*$$

During the second half of the bit-slot the receiver tests the voltage on the data line twice in succession, and if it finds that a 0-to-1 transition has occurred, it outputs a “1”. If a 1-to-0 transition is encountered, a “0” is output. If no transitions are encountered, the receiver is idle.

A summary of the events in the above expressions and their corresponding interpretation is given in *Table 1* below:

Event	Description	Event t	Description
clk_0	start of bit-slot	$clk_{1/2}$	$1/2$ bit-slot
in_{i-1}	‘1’ to be transmitted	in_{i-0}	‘0’ to be transmitted
out_{i-1}	‘1’ received	out_{i-1}	‘0’ received
$sns1_{iL}$	sense “low” on data line prior to transmission	$sns1_{iH}$	sense “high” on data line prior to transmission
$sns2_{iL}$	sense “low” on data line after transmission	$sns2_{iH}$	sense “high” on data line after transmission
d_H	output “high” on data line	d_L	output “low” on data line
rtx_i	retransmit	$idle_i$	TX/RX #i is idle

Table 1: Sender/Receiver events

The overall behavior of the protocol involving multiple senders and receivers is given by the following expression, which reflects the fact that the senders and receivers operate independently of each other:

$$B = \otimes_{i=1..n} B_{TX_i} \otimes_{j=1..m} B_{RX_j}$$

In particular, modeling the protocol with two senders and one receiver, described in [20], can be done with the following expression:

$$B = B_{TX_1} \otimes B_{TX_2} \otimes B_{RX}$$

4. FORMAL VERIFICATION OF THE DISTRIBUTED TRANSMISSION PROTOCOL

To verify the correct behavior of a communication protocol, we must guarantee that several properties of the protocol are always satisfied. Examples of such properties include:

Property 1:

If a single sender and receiver are involved in the communication and the sender sends a '1'/'0', the receiver receives a '1'/'0'.

Property 2:

If multiple senders begin transmitting simultaneously, one of the messages is received. The other message must be retransmitted.

Let us model these properties in the context of the distributed protocol involving two senders and a single receiver. Once again, the behavior of that protocol is given by the expression:

$$B = B_{TX_1} \otimes B_{TX_2} \otimes B_{RX}$$

where:

$$B_{TX_1} = (clk_0 \cdot (in_{1_1} \cdot (sns1_{1L} \cdot clk_{1/2} \cdot d_H + sns1_{1H} \cdot d_L \cdot (sns2_{1L} \cdot clk_{1/2} \cdot d_H + sns2_{1H} \cdot rtx_1)) + in_{1_0} \cdot (sns1_{1H} + sns1_{1L} \cdot d_H) \cdot clk_{1/2} \cdot d_L) + idle_{TX_1})^*$$

$$B_{TX_2} = (clk_0 \cdot (in_{2_1} \cdot (sns1_{2L} \cdot clk_{1/2} \cdot d_H + sns1_{2H} \cdot d_L \cdot (sns2_{2L} \cdot clk_{1/2} \cdot d_H + sns2_{2H} \cdot rtx_2)) + in_{2_0} \cdot (sns1_{2H} + sns1_{2L} \cdot d_H) \cdot clk_{1/2} \cdot d_L) + idle_{TX_2})^*$$

$$B_{RX} = (clk_0 \cdot clk_{1/2} \cdot (sns1_L \cdot sns2_H \cdot out_1 + sns1_H \cdot sns2_L \cdot out_0 + sns1_L \cdot sns2_L \cdot idle_{RX}))^*$$

Modeling the first property is fairly straight-forward: in any iteration of the system, if a bit '1' is transmitted, it must be followed by the receipt of a '1', and if a bit '0' is sent, it must be followed by the receipt of a '0':

$$P_1 = (in_{1_1} \cdot out_1 + in_{1_0} \cdot out_0)^*$$

The *-operator indicates that this condition must be true in any iteration.

The second property can be broken down into two sub-properties: "One of the senders transmits its complete message" and "The other sender detects a bus collision and issues a retransmit signal".

Notice that if both senders were to simultaneously transmit the same stream of bits⁴, the receiver would receive the correct message, and no collision detection is required. Collision detection is only necessary if one of the senders transmits a '1' and the other a '0'. Recall that our protocol specifies that the sender transmitting the bit '1' should detect the collision at the beginning of the bit slot, stop transmitting, and issue a retransmit signal. The sender transmitting a bit '0' will not detect the collision and, hence, continue transmitting the rest of its message. In other words, if both senders send a '0' or a '1', the receiver should receive a '0' or a '1'. If one sender sends a '0' and the other one sends a '1', then the receiver should receive a '0'. Therefore, the first part of property 2 can be expressed by the following series-parallel poset expression:

$$P_{2,1} = ((in_{1_0} \otimes in_{2_0}) \cdot out_0 + (in_{1_0} \otimes in_{2_1}) \cdot out_0 + (in_{1_1} \otimes in_{2_0}) \cdot out_0 + (in_{1_1} \otimes in_{2_1}) \cdot out_0)^*$$

The second part of property 2 – that the sender transmitting a '1' detects a collision and issues a *retransmit* can be modeled directly:

$$P_{2,2} = (in_{1_1} \cdot sns1_H \cdot d_L \cdot sns2_H \cdot rtx)^*$$

The verification of these and a number of other properties was done with the help of the SPPV Verification Environment. In the next section discuss the results of the verification and compare the modeling and verification presented above to the work in [20, 21].

5. COMPARISON WITH OTHER MODELING AND VERIFICATION APPROACHES

The verification of the Philips audio control protocol was first presented in [21]. The paper, takes an automata based approach to the verification of the protocol, but deals with the restricted case of a single sender, single receiver involved in the transaction. The work presented in [20] is an extension of the material

⁴ This is not possible, of course, since each message transmitted begins with the sender's unique address which has the prefix property.

presented in [21], and deals with the case of two senders and a single receiver. The issue of collision detection, therefore, becomes important. The verification process in [20] is also automata-based: a separate automaton is modeled for each of the two senders and the receiver, as well as for the wire, the message generator, and the checker of transmission/receipt correctness. The automata are very complex, and model all possible timing issues involved in the protocol. Due to the high complexity of the model, the verification process reportedly took over 9 hours. Extending the automata-based model in [20] to more than two senders and one receiver is not trivial, and the increased complexity of the model significantly impacts the verification time.

Compared to the work presented in [20], our model is much more general in nature: It provides the framework for the verification of the communication among multiple senders and multiple receivers (in a broadcast environment). Extending our model is a trivial matter of adding the appropriately indexed sender and/or receiver expressions for each additional sender and/or receiver in the system. Although our model does not describe all possible timing issues as does the model in [20], it represents the behavior of the protocol with accuracy sufficient for verification. A model of the protocol, reflecting all possible timing issues, is currently under development, and is a fairly straightforward extension of the current protocol model. The verification of our current model was performed in less than 10 minutes on a PC with an Intel Pentium IV 2GHz CPU and 512Mb of main memory. We expect that the verification of the new, more detailed model will not take significantly more time due to the nature of the verification methodology and the fact that the verification properties will remain the same.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we presented a formal model of a distributed transmission protocol based on the popular Philips audio control protocol. The model is expressed in the formalism of series-parallel posets, and verified using the SPPV Formal Verification environment. The verification process is significantly faster than similar verification endeavors employing other formalisms (automata-based verification), and the model is easily extended to any number of senders and receivers with

little penalty in terms of increased computation time of the verification process.

Current work is aimed at enhancing the present protocol model by adding specific timing constraints, as well as exploring how incorrect assumptions, adopted in the transmission protocol and leading to incorrect system behavior, can be detected by the series-parallel poset verification process.

REFERENCES

- [1] K. McMillan, "*Symbolic Model Checking*", Kluwer, 1993
- [2] R. Kurshan, "*Computer Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*", Princeton Series in Computer Science, Princeton, 1994
- [3] M. Nielsen, G. Plotkin, and G. Winskel, "*Petri nets, event structures, and domains*", part I, TCS, 13:85-108, 1981
- [4] V. Pratt, "*Modeling Concurrency with Partial Orders*", International Journal of Parallel Programming, 1986
- [5] P. Godefroid, "*Partial Order Methods for the Verification of Concurrent Systems: an Approach to the State Explosion Problem*", Doctoral Dissertation, University of Liege, 1995
- [6] R. Nalumasu, G. Gopalakrishnan, "*A New Partial Order Reduction Algorithm for Concurrent System Verification*", Proc of IFIP, 1996
- [7] D. Peled, "*Combining Partial Order Reductions with On-the-Fly Model Checking*", Journal of Formal Methods in Systems Design, 8 (1), 1996
- [8] L. Ivanov, R. Nunna, S. Bloom, "*Modeling and Analysis of Non-Iterated Systems: An Approach Based on Series-Parallel Posets*", Proceedings of ISCAS'99, 1999
- [9] L. Ivanov, R. Nunna, "*Formal Verification with Series-Parallel Posets of Globally-Iterated Locally-Non-Iterated Systems*", Proceedings of MWSCAS'99, 1999
- [10] L. Ivanov, R. Nunna, "*Formal Verification: A New Partial Order Approach*", Proceedings of ASIC/SOC'99, 1999
- [11] L. Ivanov, R. Nunna, "*Modeling and Verification of an Interconnect Bus Protocol*", Proc. of MWSCAS'00, 2000

- [12] L.Ivanov, R.Nunna "*Modeling and Verification of Cache Coherence Protocols*", ISCAS'01, Sydney, Australia, 2001
- [13] L.Ivanov, R.Nunna "*Modeling and Verification of Iterated Systems and Protocols*", MWSCAS'01, Dayton, OH, 2001
- [14] L.Ivanov "*Formal Verification of a Microprocessor Control*", MWSCAS'01, Dayton, OH, 2001
- [15] L.Ivanov "*Formal Verification of Microinstruction Sequencing*", Proceedings of ICCIT'01, Montclair, NJ, 2001
- [16] L.Ivanov "*Modeling and Verification of a Pipelined CPU*" MWSCAS'02, Tulsa, OK, 2002
- [17] L.Ivanov "*SPPV: A New Formal Verification Environment*" MWSCAS'02, Tulsa, OK, 2002
- [18] L.Ivanov "*Automatic Extraction of System Behavior from Verilog Specifications*", VLSI'04, Las Vegas, NV 2004
- [19] S. Bloom, Z. Esik, "*Free Shuffle Algebras in Language Varieties*", TCS 163 (1996) 55-98, Elsevier
- [20] J. Bengtsson et al., "*Verification of an Audio Protocol with Bus Collision Using UPPAAL*", CAV'96, New Brunswick, NJ, 1996
- [21] P.-H. Ho and H. Wong-Toi. "*Automated Analysis of an Audio Control Protocol*", CAV'95, #939, LNCS, 1995