An application of abstraction and induction techniques to degenerating systems of processes

A. Miller and M. Calder Department of Computing Science
University of Glasgow
Glasgow, UK
Email: alice,muffy@dcs.gla.ac.uk

***Abstract*— The major problem with model checking concurrent programs is the inability to prove that properties** *scale up*: **do results that hold for all systems of size n say, hold for a system of size N, where $N > n$? This problem is known as the** *parameterised model checking problem* **(PMCP) which is in general undecidable [1]. Model checking alone is unable to satisfy the need to answer questions about systems of any size. But it is essential that techniques for scaling and generalising results are available. In this paper we develop two approaces to generalisation (one based on abstraction, the other on induction) and illustrate their applicability in the domain of software verification, specifically in protocol analysis.**

I. Introduction

In this paper we discuss methods for generalising model checking results to systems of any size. Attempts to overcome the PMCP are not new. For example, in [2] a classification of systems into subclasses for which methods are available for generalisation is suggested and examples given. One of the most widely used approaches to tackling the PMCP when processes of a system are *isomorphic* (that is, they are identical up to labelling) is the synthesis of network *invariants* [3]–[6]. In [7] a similar approach is used to show how certain properties of a *modified* version of the tree identifition stage of the IEEE FireWire protocol (the MTIP) can be proven for any size of network – provided that the processes have a star topology. This approach involves a combination of model-checking and the construction of an *Abstract* process.

In this paper we consider two ways to generalise properties of the tree identification stage of the IEEE FireWire protocol (TIP), for any size of network. In section II we briefly describe the TIP and in section III we summarise previous attempts to formalise the behaviour of the TIP, for a fixed size of network. In section IV we show how abstraction can be applied when there is a star topology. The major contribution of this paper is given in section V in which we introduce a novel approach to generalisation, based on induction, which exploits the degenerative behaviour of the protocol – processes eventually terminate and play no further part in the protocol. We use our method to infer properties of the TIP for *any* size of network, with any topology.

Model checking plays an essential part in our proofs. Our abstraction technique relies on the verification of a fixed *Abstract* process using model cheking. In our induction technique, not only does model checking enable us to establish a *base case* for our argument but it ensures that the model upon which our proof is based is reasonable.

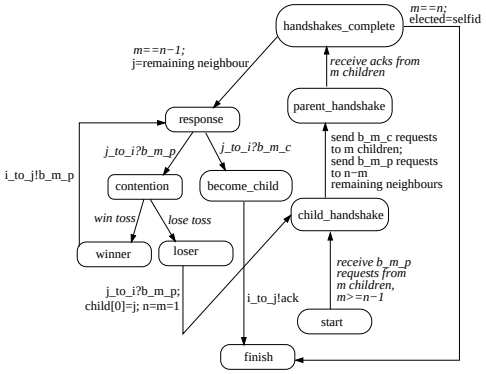

Fig. 1. The tree Identify protocol for process i

II. The tree identify phase of the IEEE tree identify protocol (TIP)

In figure 1 we give a diagrammatic representation of the automaton for a node process (Node$[i]$) communicating via the tree identify protocol (TIP). For a specific node, n denotes the number of neighbours of the node. The messages that are sent during the protocol are *be_my_parent*, *be_my_child* and *ack*. A process can not progress to the *child_handshake* state until it has received at least $n-1$ *be_my_parent* requests. (Hence a leaf node is immediately able to progress). Contention occurs when a pair of nodes send *be_my_parent* requests to each other at the same time. Figure 2 illustrates the communication sequence for one possible exectution of the TIP for a specific, acyclic configuration on 6 nodes. For convenience abbreviations of the messages are used in the figures. Note that in figure 2, $Node[4]$ is the first to send an acknowledgement and terminate. The size of the network decreases as the protocol progresses. This is an important observation and will be discussed further in section V.

III. Formal approaches to verifying the TIP

The TIP has been specified via a variety of different formalisms, e.g. E-LOTOS [8], I/O automata [9], [10] and μCTL [11]. For a summary of specification only approaches see [12]. Verification of the TIP for fixed confugurations of processes has been achieved via model-checking (using SPIN

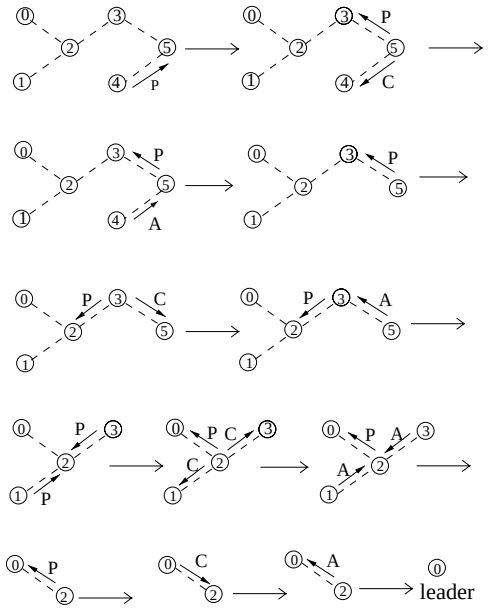

Fig. 2. Example behaviour of TIP on 6 nodes

[7] and SMV [13] for example) and thorough analysis of the RCP achieved using probablistic techniques (see section III-A below). Many of these and other verification approaches appear in [14] together with a full comparative case study.

A. Root Contention

The root contention protocol (RCP) is an important sub-protocol of the TIP. The RCP involves complicated timing parameters and the random selection of waiting periods. Thus in order to model this part of the protocol realistically, notions of real-time, randomization and timing must be taken into consideration. Attempts to model check the TIP using real-time have concentrated solely on the RCP, leaving the verification of the rest of the TIP for non real-time methods (see section III-B below). As only two nodes are ever involved in contention, verification of the RCP only involves considering the parallel composition of two processes - unlike the verification of the whole TIP, where fixed networks of a given size are considered.

Various attempts have been made to verify the RCP for a fixed size of network [15]–[17]. In the more general case, Kwiatowska et al [18] use probabilistic timed automata based on classic timed automata [16] to study deadline properties of the RCP. Finite-state Markov decision processes are obtained from the automata via a property-preserving discrete-time semantics. The minimal probablilty that RCP elects a leader within a given deadline is then computed. The results are highly convincing – given a long enough deadline contention will eventually be resolved. (The probability of electing a leader converges to 1 as the deadline approches ∞). This is equivalent to proving that contention will always eventually be resolved.

B. Model checking a fixed configuration of node processes

Model checking is an automatic technique for verifying finite state concurrent systems. Systems are specified using a modelling language and the Kripke structure [19] associated with this specification checked to verify given temporal properties.

Definition 1: Let AP be a set of atomic propositions. A Kripke structure over AP is a tuple $\mathcal{M} = (S, S_0, R, L)$ where S is a finite set of states, S_0 is the set of initial states, $R \subseteq S \times S$ is a transition relation and $L : S \to 2^{AP}$ is a function that labels each state with the set of atomic propositions true in that state.

In [7] a set of properties of the TIP are verified using SPIN [20]. SPIN is a generic verification system that supports the verification of asynchronous process systems. A specification is written in the verification language Promela (PROcess MEta LAnguage) and correctness claims specified in the syntax of linear temporal logic (LTL). The Promela description of the concurrent system consists of one or more process templates or *proctype* definitions plus a process instantiation. Each template is translated by SPIN into a finite automaton and the global behaviour of the concurrent system is obtained by computing the asynchrounous interleaving product of the automata. This interleaving product is often referred to as the state space of the system (or global reachability graph). We do not give details here of how SPIN performs LTL model checking (see [20]). For our purposes it is sufficient to note that verification of a property ϕ for a given Promela specification using SPIN implies that ϕ holds for the Kripke structure representing (the state graph of) our Promela specification.

The model of the TIP described in [7] consists of a parameterised specification of a Node process p say, based on the description of the protocol given in figure 1. Communication is asynchronous and takes place via channels of length 1. This is a good reflection of the physical case - wires only convey one "message" in a given direction at a time. (SPIN) verification of a given property ϕ entails running a number (N) of instantiations of p (with a chosen configuration, C say) concurrently against an LTL formula representing ϕ. Successful verification implies that ϕ holds for the specific configuration C on N processes.

In the model described in [7] contention is modelled as a simple coin toss (in which the first process to reach the *contention* state wins the toss). There is no concept of real-time in SPIN, although the notions of *always* and *eventually* can be incorporated into the LTL properties via the $[]$ and $\langle\rangle$ operators. We can only assert that contention is *eventually* resolved – not that it is resolved within a given time-limit. Because of the time parameters and random selection of waiting times that is involved in the RCP it is not possible to model contention realistically within our framework. Therefore we leave the proof of the RCP to the real-time/probablilistic real-time community (see section III-A above) who have proved [18] that contention will eventually be resolved, and work under this assumption.

A set of properties were checked for all configurations of N nodes, where $2 \leq N \leq 6$. The Promela description

varies depending on the configuration and the property to be verified. An example Promela model for the TIP for a given acyclic configuration on 6 nodes can be found on our website [21]. Four of the properties with their LTL descriptions are given below. The LTL descriptions rely on a variable ($elected$) that appears in the Promela description. This variable is used to record the id of the current leader. The default value of $elected$ is N, the number of processes (6 in our case) - the ids of the processes taking values between 0 and $N-1$. The proposition $(node[proci]@start)$ states that node with id i is at the label $start$ (i.e. reached the $start$ state). All properties which contain a free variable (i) are said to be *indexed by* i and must be verified for all instantiations i' of i, $0 \leq i' \leq 5$.

Property 1 *(Under the assumption of weak-fairness) process i will not wait for a "be_my_parent" request for an infinitely long time.*
That is $[](\langle\rangle\neg p)$ where p is $(node[proci]@start)$.

Property 2 *A leader will always be elected.*
That is $(\langle\rangle q)$ where q is $\neg(elected == N)$.

Property 3 *(Under the assumption of weak-fairness) it is possible for process i to be elected leader.*
That is $\neg\langle\rangle r$ is violated, where r is $(elected == i)$.

Property 4 *Only one process will be elected leader.*
That is $[](p \rightarrow ([]p))$ where p is $(elected == i)$.

Property 1 represents the loop detection aspect of the TIP (which we do not describe here) and holds for all acyclic configurations and no cyclic configurations. Weak fairness is a feature of the type of paths that are to be considered. We are only interested in paths for which no enabled process fails to make a transition indefinitely. SPIN has a *weak fairness* option which restricts a search only to the paths of interest. Properties 2–4 hold for all acyclic configurations, for all $2 \leq i \leq 6$. (See [7] for experimental results of SPIN verification for example configurations on 6 nodes.)

IV. Using abstraction to verify properties of the TIP for a star topology of any size

We are now ready to discuss generalisation.

In this section we consider star configurations of processes following the TIP. That is, networks consisting of a central node process $Central\_node(0)$ say, which is connected to $N-1$ other leaf nodes $Leaf[1], Leaf[2], \ldots, Leaf[N-1]$. First we discuss a simplified star model for a fixed number of processes. Our abstraction will be based on this simplified model.

A. A simplified star model for fixed N

It is fairly straightforward to create a model of the TIP for such a configuration for a fixed value of N using Promela. Indeed one could use the general Promela model of the TIP given in [7]. However, for simplicity we use a less general description (specific to star topologies) consisting of an instantiation of a $Central\_node$ proctype (with $id = 0$) together with $N-1$ intanitations of a $Leaf$ proctype. We can use this model to verify any of the properties described in section III-B for any small, fixed number of processes. An example of such a model (a *simplified star model*) used to verify property 4 with $i = 1$ for a 6-node star configuration can be found on our website [21].

In the simplified star model the "coin toss" procedure is less complicated because communication only occurs between the $Central\_node$ process and the $Leaf$ processes (not between the $Leaf$ processes themselves).

B. The abstracted star configuration

In this section we introduce a technique based on abstraction and induction to prove safety properties of the TIP (for example our property 4) for a star topology for any number of processes. One can not hope to prove the properties using induction alone [4], [6], because the behaviour of the $Central\_node$ process depends upon the total number of processes. However, an important observation is that for a star topology consisting of N processes, much of the behaviour of the $Central\_node$ processes depends upon whether the number of messages that have at arrived at its ports (channels from neighbours) is less than $N-2$, equal to $N-2$ or equal to $N-1$, rather than the specific number of messages.

Another important observation is that in order to verify property depending only on process i, the internal behaviour of all processes other than i can be ignored. Assume for the time being that $i \neq 0$. Without loss of generality, we may assume that $i = 1$. Our abstraction approach involves modelling $Leaf[1]$ as before and modifying the communication between $Central\_node[0]$ and all leaf nodes with $id \neq 1$. We refer to leaf processes with ids $\neq 0$ or 1 as *abstract* processes. The modified $Central\_node$ process, $Central\_node'[0]$ communicates with $Leaf[1]$ as before. However, instead of reading messages from an abstract proces, $Central\_node'[0]$ makes a non-deterministic choice over the set of messages (or whether the number of messages is less than $N-2$, equal to $N-2$, or equal to $N-1$) which may (or may not) have arrived at that point. Unlike $Central\_node[0]$, $Central\_node'[0]$ does not write messages to any abstract process, apart from sending an ack message to an abstract process when $Central\_node'[0]$ has become a child of an abstract process. In all other instances $Central\_node'[0]$ makes a dummy move (*skip*) and no message is sent. Thus all communication between $Central\_node'[0]$ and abstract processes, apart from the sending of ack from $Central\_node'[0]$ to an abstract process, can be thought of as being *virtual*. Our approach is summarised in figure 3. $Central\_node[0]$ and $Central\_node'[0]$ are represented by q and q' and, for $1 \leq i \leq N-1$, p_i represents $Leaf[i]$. The cloud labelled *Abstract* represents all of the abstract processes. Solid arrows represent communication between processes and dotted arrows represent virtual communication.

Another feature of the abstracted (N-process) model is that contention between the $Central\_node$ process and any abstract process now involves a non-deterministic choice over the result of a coin toss.

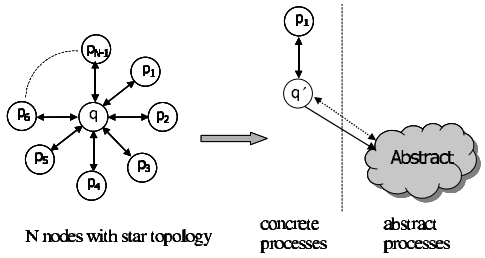

Fig. 3. Abstraction technique for N-process model, star topology

Let us refer to (the Kripke structure)

$$\mathcal{M}_N = \mathcal{M}(Central\_node||Leaf[1]||\ldots||Leaf[N-1])$$

as the concrete model of size N. Note that in figure 3 processes p_1 and q' are referred to as *concrete processes*. This is because they are, like the processes of the concrete model, fixed, finite processes. (The abstract process is also fixed and finite, but *represents* the behaviour of any number of processes.)

Let us define

$$\mathcal{M}_{Abs_1} = \mathcal{M}(Central\_node'||Leaf[1]||Abstract)$$

to be the $abstracted_1$ model. Correspondingly we can define a family of models $\{\mathcal{M}_{Abs_i}\}_{i=0}^{N-1}$ – the $abstracted\_i$ models – where, for $1 \leq i \leq N-1$,

$$Abs_i = Central\_node'||Leaf[i]||Abstract$$

and

$$Abs_0 = Central\_node'||Abstract.$$

We show, using abstraction that properties that hold for the $abstracted\_i$ models hold for all concrete models of size N, for all N greater than a specified minimal size. That is, we show

Theorem 1: If ϕ_i is a property indexed by i and, for all $i' \geq 0$ let $\phi_{i'}$ be the instantiation of ϕ_i with $i = i'$, then, if $\mathcal{M}_N$ is the concrete model (for a star topology) of size N,

1) ff $\mathcal{M}(Abs_0) \models \phi_0$ then $\mathcal{M}_N \models \phi_0$ for all $N \geq 3$ and
2) If $\mathcal{M}(Abs_1) \models \phi_1$ then $\mathcal{M}_N \models \phi_{i'}$ for all $i' \geq 1$, for all $N \geq min(i', 4)$.

In particular,

Corollary 1: If ϕ_i is property 4 then

1) $\mathcal{M}_N \models \phi_0$ for all $N \geq 3$ and,
2) for all $i \geq 1$, $\mathcal{M}_N \models \phi_i$ for all $N \geq max(i, 4)$.

We can not give a full proof of theorem 1 here, for space reasons. However below we give a brief outline of the proof.

In order to prove Theorem 1 we first define two reduced forms of the model $\mathcal{M}_N$, namely $\mathcal{M}_N^{r_i}$ and $\mathcal{M}_N^{r_0}$. These reduced structures are constructed via data abstraction [22] and (by a result proved in [23]), any property that can be shown to hold for a reduced model, holds for the concrete model. It can be shown that for any $N \geq 3$ there is a simulation preorder [24] between $\mathcal{M}^{r_0}$ and $\mathcal{M}_{Abs_0}$. Similarly, for all $i \geq 1$, for all $N \geq max(i, 4)$, there is a simulation preorder between $\mathcal{M}^{r_i}$ and $\mathcal{M}_{Abs_i}$. For models $\mathcal{M}$ and $\mathcal{M}'$, $\mathcal{M}' \models \phi$ implies that $\mathcal{M} \models \phi$. Proof of theorem 1 follows.

When ϕ_i is property 4, we can show, using model checking, that $\mathcal{M}_{Abs_0} \models \phi_0$ and $\mathcal{M}_{Abs_1} \models \phi_1$. In addition we can prove that, in this case, $\mathcal{M}_{Abs_1} \models \phi_1$ if and only if $\mathcal{M}_{Abs_i} \models \phi_i$, for all $i > 1$. Hence Corollary 1 follows from theorem 1.

C. Limitations of this approach

We can only use this abstraction when we have a star topology. In addition, we can only prove properties that *hold for the abstracted model*. Due to the additional non-determinism in the abstracted model, many properties that should hold for a concrete model of any size can not be shown to do so. For example, property 2 does not hold in the abstracted model, so can not be shown to hold for the concrete model using this approach.

V. Using Induction to exploit the degenerative nature of the TIP

In this section we use the degenerative behaviour of the TIP together with induction (on network size) to prove the following theorem. Here $\mathcal{M}_{\Gamma_n}$ is the model of a network of n processes with configuration Γ_n.

Theorem 2: If ϕ is a property that is not indexed by any process id and $\mathcal{M}_N$ is a model of a network of N processes following the TIP, then if $\mathcal{M}_{\Gamma_{n_0}} \models \phi$, for all configurations Γ_n for all $n \geq n_0$, for some $2 \leq n_0 \leq N$, and if $\mathcal{M}_{\Gamma_n} \models \phi$ for all star configurations Γ_n where $n \leq N$, $M_N \models \phi$.

It is easy to prove some properties for a star topology of any size. For example, if ϕ is property 2 we can show that, for any N, if $\mathcal{M}_N$ is a model relating to a star topology, $\mathcal{M}_N \models \phi$ for any $n \geq 2$. The paths (of states) relating to such a model can be divided into $3(N-1)+1$ types. Assuming $Node[0]$ is the central node the types are: $Type_0$ and $Type_{j,A}$, $Type_{j,B}$ and $Type_{j,C}$, for $1 \leq j \leq N-1$. In $Type_0$ paths, $Node[0]$ receives $N-1$ *be_my_parent* requests, sends *be_my_child* requests to all $N-1$ children, recieves *ack* responses from all children and becomes leader.

In $Type_{j,A}$, $Type_{j,B}$ and $Type_{j,C}$ paths, $Node[0]$ receives *be_my_parent* requests from all neighbours apart from $Node[j]$, and sends *be_my_child* requests to its $N-2$ children. The difference between these 3 sets of path is if and when $Node[j]$ sends a *be_my_parent* request and, if contention arises, the result of the coin toss.

Considering each type of path in detail, it can be shown that property 2 holds along every type of path, and hence for all paths. Let us assume therefore that we have an acyclic topology that is not a star. Below we give a brief outline of the proof of theorem 2 for such a topology.

For any topology (network configuration or graph) Γ_N on N nodes, we say that a node, $Node[i]$ say, is a *level* 1 node if $Node[i]$ is not a leaf node and $Node[i]$ has precisely one neighbour that is not a leaf node.

Using basic graph theory one can show that, if Γ_N is acyclic and not a star, then Γ_N has at least one level 1 node. If Γ_N is not a star, for all paths in $\mathcal{M}_N$, some level 1 node must be the

first to reach the *child_handshake* state (see figure 1), and have received *be_my_parent* requests from all of its leaf neighbours. For every level 1 node, $Node[j]$, let us define $clip_j(\Gamma_N)$ to be the graph formed by removing all of the leaf neighbours of $Node[j]$ from Γ_N and $\mathcal{M}_{clip_j(\Gamma_N)}$ the corresponding model.

Let AP be the set of atomic propositions of $\mathcal{M}_N$. It can be shown that every path in which $Node[j]$ is the first to reach *child_handshake* is stuttering equivalent [25], with respect to a reduced set of atomic propositions $AP' \subset AP$ to a path in $\mathcal{M}(clip_j(\Gamma_N))$. The set AP' does not include propositions relating to the channels between $Node[j]$ and its leaf neighbours or to variables associated with the leaf neighbours of $Node[j]$. This is an important point, and precludes properties which are indexed by process ids from the theorem.

It follows that if ϕ is invariant under stuttering and, for all level 1 nodes $Node[j]$, $\mathcal{M}(clip_j(\Gamma_N)) \models \phi$, then $\mathcal{M}_N \models \phi$. Now, for all level 1 nodes $Node[j]$, $clip_j(\Gamma_N)$ is either a star, in which case $\mathcal{M}(clip_j(\Gamma_N)) \models \phi$ (by the assumption of the theorem), or is not a star, and so has at least one level 1 node. In this case our previous argument applies and the graph can be further reduced. Continuing in this way, we continue to generate smaller graphs, such that the validity of ϕ for the corresponding models implies validity of ϕ for $\mathcal{M}_N$. In all cases, we eventually arive at a star or a graph Γ_n of size $n \leq n_0$ in which case validity of ϕ is assumed, and theorem 2 follows.

A. Limitations of this approach

The major limitation of this type of generalisation is that it applies to very specific types of systems, namely degenerating systems. We can only prove properties that are invariant under stuttering. (Since none of our properties involve the next time operator (X) they are all invariant under stuttering). As theorem 2 only applies to properties that do not depend on any process id, of our properties property 2 is the only property to which this approach applies. However, as all of the conditions of theorem 2 hold in this case (we have shown that property 2 holds for all acyclic networks of size ≤ 6 and for star topologies of any size). Hence property 2 holds for all acyclic networks of any size (> 2).

VI. Conclusions

We have proved some important generalisation results for a well-known, ubiquitous protocol. We have applied currently known techniques to prove one generalisation result using abstraction, and exploited the degenerative nature of the protocol to prove another using a novel application of induction.

Whilst the results are largely specific to the TIP, we believe that the methods used can be widely applied to other domains.

References

[1] K. R. Apt and D. C. Kozen, "Limits for automatic verification of finite-state concurrent systems," *Information Processing Letters*, vol. 22, pp. 307–309, 1986.

[2] M. Calder and A. Miller, "Five ways to use induction and symmetry in the verification of networks of processes by model-checking," in *Proceedings of the Second Workshop on Automated Verification of Critical Systems (AVoCS 2002)*, 2002, pp. 29–42.

[3] E. Clarke, O. Grumberg, and S. Jha, "Verifying parameterized networks using abstraction and regular languages," in *[26]*, 1995, pp. 395–407.

[4] R. P. Kurshan and K. McMillan, "A structural induction theorem for processes," in *Proceedings of the eighth Annual ACM Symposium on Principles of Distrubuted Computing*. ACM Press, 1989, pp. 239–247.

[5] P. Wolper and V. Lovinfosse, "Properties of large sets of processes with network invariants (extended abstract)," in *[27]*, 1989, pp. 68–80.

[6] M. Browne, E. Clarke, and O. Grumberg, "Reasoning about networks with many identical finite state processes," *Information and Computation*, vol. 81, pp. 13–31, 1989.

[7] M. Calder and A. Miller, "Using SPIN to analyse the tree identification phase of the IEEE 1394 high performance serial bus (FireWire) protocol," in *[14]*. Springer-Verlag, 2003, pp. 247–266.

[8] C. Shankland and A. Verdejo, "Time, E-LOTOS and the FireWire," in *Proceedings of the Workshop on Formal Methods and Telecommunications*, M. Marsane, J. Quemada, T. Robles, and M. Silva, Eds., Prensas Universitarias de Zaragoza, 1999, pp. 103–119.

[9] M. Devillers, W. Griffioen, J. Romijn, and F. Vaandrager, "Verification of a leader election protocol – formal methods applied to IEEE 1394," *Formal Methods in System Design*, vol. 16, no. 3, pp. 307–320, 2000.

[10] J. Romijn, "A timed verification of the IEEE 1394 leader election protocol," *Formal Methods in System Design*, vol. 19, no. 2, pp. 165–194, 2001, special issue on FIMACS'99.

[11] C. Shankland and M. van der Zwaag, "The tree idnetify protocol of IEEE 1394 in μCTL," *Formal Aspects of Computing*, vol. 10, pp. 509–531, 1998.

[12] S. Maharaj and C. Shankland, "A survey of formal methods applied to leader election in IEEE 1394," *Journal of Universal Computer Science*, vol. 6, no. 11, pp. 1145–1163, October 2000.

[13] V. Schuppen and A. Biere, "Verifying the IEEE 1394 tree identify protocol with SMV," in *[14]*. Springer-Verlag, 2003.

[14] J. Maharaj, S. Romijn and C. Shankland, Eds., *Formal Aspects of Computing: Special issue on IEEE 1394 Tree Identification Protocol*. Springer-Verlag, 2003, vol. 14, no. 3.

[15] M. Stoelinga and F. Vaandrager, "Root contention in IEEE 1394," in *[28]*, 1999, pp. 53–74.

[16] S. Simons and M. Stoelinga, "Mechanical verification of the IEEE 1394a root contention protocol using UPPAAL2k," *Software Tools for Technology Transfer*, vol. 3, no. 4, pp. 469–485, 2001.

[17] K. Larson, P. Pettersson, and W. Yi, "UPAAL in a nutshell," *Software Tools for Technology Transfer*, vol. 1, no. 1+2, pp. 134–152, 1997.

[18] M. Kwiatkowska, G. Norman, and J. Sproston, "Probalistic model checking of deadline properties in the the IEEE 1394 FireWire root contention protocol," in *[14]*. Springer-Verlag, 2003.

[19] E. M. Clarke, O. Grumberg, and D. Peled, *Model Checking*. The MIT Press, 1999.

[20] G. J. Holzmann, "The model checker Spin," *IEEE Transactions on Software Engineering*, vol. 23, no. 5, pp. 279–295, May 1997.

[21] M. Calder and A. Miller, "Veriscope publications website: `http://www.dcs.gla.ac.uk/research/veriscope/publications.html`."

[22] E. Clarke, O. Grumberg, and D. Long, "Model checking and abstraction," *ACM Transactions on Programming Languages and Systems*, vol. 16, no. 5, pp. 1512–1542, January 1994.

[23] ——, "Model checking and abstraction," in *in [29]*, 1992, pp. 343–354.

[24] R. Milner, "An algebraic definition of simulation between programs," in *Proceedings of the 2nd International Joint Conference on Artificial Intelligence*, 1971, pp. 481–489.

[25] L. Lamport, "What good is temporal logic?" *Information Processing*, vol. 83, pp. 657–668, 1983.

[26] I. Lee and S. A. Smolka, Eds., *Proceedings of the 6th International Conference on Concurrency Theory (CONCUR '95)*, ser. Lecture Notes in Computer Science, vol. 962. Philadelphia, PA.: Springer-Verlag, August 1995.

[27] J. Sifakis, Ed., *Proceedings of the International Workshop in Automatic Verification Methods for Finite State Systems*, ser. Lecture Notes in Computer Science, vol. 407. Grenoble, France: Springer-Verlag, June 1989.

[28] J. Katoen, Ed., *Formal Methods for Real-Time and Probabilistic Systems. Proceedings of the 5th International AMAST Workshop (ARTS'99)*, ser. Lecture Notes in Computer Science, vol. 1601. Bamberg, Germany,: Springer-Verlag, May 1999.

[29] *Conference Record of the Nineteenth Annual ACM Symposium on Principles of Programming Languages (POPL '92)*. Albuquerque, New Mexico: ACM Press, January 1992.