Simple Representative Instantiations for Multicast Protocols

Javier Esparza and Monika Maidl

School of Informatics, University of Edinburgh
{jav+monika}@inf.ed.ac.uk

Abstract. We present a formal model for multicast network protocols working on arbitrary tree structures. We give sufficient conditions under which correctness of the protocol for all structures reduces to correctness for the structures with at most one layer of internal nodes. If additional conditions hold, we can reduce further to correctness for one single structure. All these results can be applied to (an abstract version of) the Pragmatic General Multicast protocol.

In the last years, much effort has been devoted to the verification of parameterised distributed systems, i.e., distributed systems designed to work correctly independently of the number of processes taking part in them. Classical examples of these systems are distributed algorithms for leader election, byzantine agreement, or distributed termination, and communication protocols, like cache coherence or network protocols. In this paper we study multicast network protocols working on tree structures.[1] In these protocols, data are exchanged between a sender (the root of the tree) and several receivers (the leaves) via network elements (the internal nodes). Messages flowing from sender to receivers can be multicasted, i.e., simultaneously sent to several successors. Examples of such multicast network protocols are PGM (Pragmatic General Multicast) [S+00] and LMS (Light-weight Multicast Services) [PPV].

Verifying a property ϕ of a parameterised system consists of checking that ϕ for all possible structures (in our case, for all possible trees). This may be difficult, and so a common approach is to first reduce the task to checking ϕ for a restricted class of structures (see for instance [EN95]). This is also the approach of this paper.

We first provide a general formal model for multicast networks. The only assumption is that messages can overtake other messages and can get lost, but cannot be duplicated. This is a reasonable assumption for protocols in which channels are just an abstraction for a routing mechanism that may send different messages—or different fragments of the same message—through various routes, as in the PGM and LMS protocols. For protocols in which messages cannot overtake others or get lost, the assumption overapproximates the behaviour of the protocol. In this case, correctness in our model still implies correctness of

[1] Actually, these protocols run on arbitrary networks, but use a distribution tree to broadcast messages. We assume that this tree has already been established.

H. Garavel and J. Hatcliff (Eds.): TACAS 2003, LNCS 2619, pp. 128–143, 2003.

the protocol. We define a notion of simulation that preserves stuttering-invariant linear-time properties, i.e., if the simulating structure satisfies the property, then the simulated structure also satisfies it.

Equipped with this formal setting, we identify general sufficient conditions for a protocol P to be *collapsable*, meaning that a property holds for all instantiations $\mathcal{N}(P)$ of P if and only if it holds for the set of instantiations $\mathcal{N}^1(P)$ with at most one level of internal elements. We prove that for every instantiation T in $\mathcal{N}(P)$ there is an instantiation T' in $\mathcal{N}^1(P)$ that simulates T.[2] Then, all instantiations in $\mathcal{N}^1(P)$ satisfy ϕ if and only if all instantiations in $T \in \mathcal{N}(P)$ satisfy ϕ, because simulation preserves properties and $\mathcal{N}^1(P) \subseteq \mathcal{N}(P)$.

In particular, our conditions are satisfied by network elements that only perform 'forwarding', i.e., only forward messages down from the parent to all children, and up from some child to the parent. Hence, our result can be applied to telecommunication protocols that do not assume that router support can be used and that use a fixed distribution tree. We show that they are also satisfied by the PGM protocol, where network elements have a much richer functionality, which is used to make communication between the sender and receivers more reliable and efficient.

While the collapse of $\mathcal{N}(P)$ to $\mathcal{N}^1(P)$ removes the problem of dealing with different tree topologies, it still leaves us with an infinite number of possible instantiations. This cannot be avoided as long as more receivers can generate more behaviour. However, we prove that if the number of different messages that can circulate is finite and receivers and network elements can repeatedly send the same message upwards, then the verification task can be further reduced: Given a property ϕ, all instantiations of $\mathcal{N}^1(P)$ satisfy ϕ if and only if one single universal instantiation U, which depends on Φ. Again, we prove that if the number of messages in the PGM protocol is bounded, then the result can be applied, and the protocol has a universal instantiation.

The paper is structured as follows. In section 1 we introduce (a version of) the PGM protocol, in order to introduce network protocols and have a rich running example for our definitions and results. Section 2 contains our formal model. Our notions of property and simulation are given in Section 3. Section 4 presents the sufficient conditions for a network to be collapsable. This section is divided into two parts; the first part deals with the special case, in which network elements can only forward messages, and the second deals with the general case. The section also shows that the PGM protocol satisfies the conditions. Finally, section 5 presents the universal instance that can simulate any other instantiation of a collapsable protocol, assuming that the number of messages is finite. The paper is accompanied by a technical report [EM02] which contains full proofs.

[2] Notice that we always speak of instantiations *of the same protocol*. Given a protocol P, one can always find another protocol P' such that every $T \in \mathcal{N}(P)$ is simulated by some $T' \in \mathcal{N}^1(P')$ by making the sender, receiver and internal processes more complicated.

1 The PGM Protocol

The Pragmatic General Multicast (PGM) protocol is a reliable multicast protocol for the distribution of information from multiple senders to multiple receivers. It is designed in order to minimize loading of the network due to acknowledgment messages or retransmissions of lost packets, and has been presented to the Internet Engineering Task Force as an open reference specification. We consider the following abstract, untimed variant of the protocol for one sender. We have a tree of processes connected by bidirectional channels. Messages can get lost and can overtake each other (i.e., can be delivered in a different order than they are sent), but cannot be duplicated. The root of the tree is the sender, and the leaves the receivers. The other processes are internal *network elements*. The source multicasts a numbered sequence of data packets called odata(*nr*,*trl*) (for original data) within a transmit window; *nr* is the number of the package, and *trl* is the left-hand edge of the sender's window at the moment of sending it. Network elements forward these packets down the distribution tree. If a receiver detects that packet *nr* is missing from the sequence, it repeatedly sends a *primary negative acknowledgment* (pnak(*nr*)) to its parent, requesting a repair. Each network element that receives a pnak forwards it to its parent, and multicasts a nak-confirmation (ncf(*nr*)) to its children; it then keeps sending secondary nak (snak(*nr*)) to its parent (which are forwarded upwards, but do not generate confirmations) until it receives a *nak-confirmation* itself. When the source receives a pnak or snak it provides a repair (rdata(*nr*,*trl*)), which is multicasted downwards to the processes that requested them. There is a final feature called *nak-anticipation*: A receiver may receive a confirmation to a pnak sent by *another* receiver. Anticipating its own future need for a repair, it repeatedly sends snak's to its parent, until either the original data or the repair arrives. Notice that odata-, rdata- and ncf-messages travel *downwards* (from sender to receivers) while pnak- and snak-messages travel *upwards* (from receiver to sender).

Formally, the protocol is given by three agents describing the sender, the receivers, and the network elements, whose descriptions can be found in Table 1. Every agent has a set of *variables* and a set of (atomic) *transitions*. Transitions are guarded by either boolean expressions over the process variables or by the delivery of a message. In the initial state, all sets are empty, *odata*, *txw_trail* and *rxw_trail* are 0, and *WIN_SIZE* has some fixed value (window size).

Our version of the PGM protocol differs slightly from [S$^+$00] in that we distinguish between primary and secondary nak messages. While our result also holds for the original version, as shown in [Mai02], our version allows for a generic proof, and it can simulate the original version except for a behaviour which is not desirable according to the specification: Our version is more economic in the number of nak messages sent than the original one. All these points are discussed in detail in the full version [EM02] of this paper.

Table 1. Agents of the PGM protocol

agent source
odata, *WIN_SIZE*, TXWTR: $\mathbb{N}$; *rec_nak*: set of $\mathbb{N}$
s1: in pnak(nr) $\vee$ in snak(nr) $\rightarrow$ out ncf(nr) downwards;
$txw\_trail < nr \rightarrow rec\_nak := \text{add}(rec\_nak, nr)$;
s2: is_in(nr, *rec_nak*) $\rightarrow$ $nr > txw\_trail \rightarrow$ out rdata(nr, *txw_trail*) downwards;
$rec\_nak := \text{remove}(rec\_nak, nr)$;
s3: length(*rec_nak*) $= 0 \rightarrow$ out odata(*odata*, *txw_trail*) downwards;
$odata := odata + 1$;
$odata + 1 > WIN\_SIZE + txw\_trail$
$\rightarrow txw\_trail := txw\_trail + WIN\_SIZE$;
endagent;

agent network_element
set_repair: set of $\mathbb{N}$; *set_interf*: set of ($\mathbb{N}$, channel_name)
e1: in pnak(nr) $\rightarrow$ $set\_repair := \text{add}(set\_repair, nr)$;
$set\_interf := \text{add}(set\_interf, (nr, c))$; [$c$ reception channel]
pnak $\notin set\_repair \rightarrow$ out pnak(nr) upwards;
out ncf(nr) downwards;
e2: in snak(nr) $\rightarrow$ $set\_interf := \text{add}(set\_interf, (nr, c))$; [$c$ reception channel]
snak $\notin set\_repair \rightarrow$ out snak(nr) upwards;
e3: in rdata(nr, trl) $\rightarrow$ $set\_repair := \text{remove}(set\_repair, nr)$;
out rdata(nr,trl) to all channels c' s.t. $(nr, c') \in set\_interf$
$set\_interf := \text{remove}((nr, c'), set\_interf)$
e4: in ncf(nr) $\rightarrow$ $set\_repair := \text{remove}(set\_repair, nr)$;
e5: in odata(nr, trl) $\rightarrow$ out odata(nr, trl) downwards;
e6: is_in(nr, *set_repair*) $\rightarrow$ out snak(nr) upwards;
endagent;

agent receiver
rxw_trail, *set_nr*, *set_missing*: set of $\mathbb{N}$
r1: in odata(nr, trl) $\wedge$ $rxw\_trail < nr \rightarrow rxw\_trail < trl \rightarrow rxw\_trail := trl$;
$set\_nr := \text{add}(set\_nr, nr)$;
for all $(rxw\_trail < i < nr \wedge i \notin set\_nr)$
$set\_missing := \text{add}(set\_missing, i)$;
$set\_missing := \text{remove}(set\_missing, nr)$;
$set\_smissing := \text{remove}(set\_smissing, nr)$
r2: in ncf(nr) $\wedge$ $rxw\_trail < nr \wedge nr \notin set\_nr \rightarrow set\_smissing := \text{add}(set\_smissing, nr)$
for all $(rxw\_trail < i < nr \wedge i \notin set\_nr)$
$set\_smissing := \text{add}(set\_smissing, i)$;
r3: in rdata(nr, trl) $\wedge$ $rxw\_trail < nr \rightarrow rxw\_trail < trl \rightarrow rxw\_trail := trl$;
$set\_nr := \text{add}(set\_nr, nr)$;
$set\_missing := \text{remove}(set\_missing, nr)$;
$set\_smissing := \text{remove}(set\_smissing, nr)$
r4: is_in(nr, *set_missing*) $\rightarrow$ $nr > rxw\_trail \rightarrow$ out pnak (nr) upwards;
$nr \leq rxw\_trail \rightarrow set\_missing := \text{remove}(set\_missing, nr)$;
r5: is_in(nr, *set_smissing*) $\rightarrow$ $nr > rxw\_trail \rightarrow$ out snak(nr) upwards;
$nr \leq rxw\_trail \rightarrow set\_smissing := \text{remove}(set\_smissing, nr)$;
endagent;

2 A Formal Model of Network Protocols

In this section we formalise the notions of tree networks, and of families of tree networks defined by a protocol.

2.1 Messages, Actions, Events, and Histories

Let M be a (possibly infinite) set of *messages*. We assume that M contains an 'empty' message, denoted by $\perp$. We assume that $M = M{\uparrow} \cup M{\downarrow} \cup \perp$ [3] where $M{\uparrow}$ and $M{\downarrow}$ are sets of *upward* and *downward* messages such that $M{\uparrow} \cap M{\downarrow} = \emptyset$. We model receiving or sending no message as receiving or sending the "empty" message $\perp$. An *abstract action*, or just an *action*, is a triple $(i, o_1, o_2) \neq (\perp, \perp, \perp)$ of messages such that $o_1 \in M{\uparrow} \cup \perp$ and $o_2 \in M{\downarrow} \cup \perp$. Intuitively, an action models receiving a message i, and sending a message o_1 upwards and a message o_2 downwards. A *trace* is a finite sequence of actions. We denote the set of all traces by TR.

Let Ch be a set of *downward channels*, and let $ch \notin Ch$ be an *upward channel*. A *concrete action* or *event* over ch, Ch is a fivetuple (i, c, o_1, o_2, C), where (i, o_1, o_2) is an abstract action, $c \in \{ch\} \cup Ch$, $C \subseteq Ch$, and moreover, either $c = ch$ and $i \in M{\downarrow} \cup \perp$, or $c \in Ch$ and $i \in M{\uparrow} \cup \perp$. (The intuition is that ch is the channel communicating with the parent, and Ch the channels communicating with the children.) An event corresponds to a process receiving message i through channel c, sending o_1 through the upward channel ch, and sending o_2 through a subset C of downward channels. We denote the set of all events by E. A *history* is a finite sequence of events. We denote the set of all histories by H.

Given an event $e = (i, c, o_1, o_2, C)$ over ch, Ch, we define the *action corresponding to* e as (i, o_1, o_2), and denote it by $a(e)$. Given a channel c', we define the *c'-action* corresponding to e, denoted by $c'(e)$, as the pair $(c'(i), c'(o_1, o_2))$ given by: (1) $c'(i) = i$ if $c' = c$ and $c'(i) = \perp$ otherwise, and (2) $c'(o_1, o_2) = o_1$ if $c' = ch$, $c'(o_1, o_2) = o_2$ if $c' \in C$, and $c'(o_1, o_2) = \perp$ otherwise.

Given a history $h = e_1 \dots e_n$, we define its *associated trace* as the sequence $tr(h) = a(e_1) \dots a(e_n)$. Given a channel c, at most one message is sent through c during an action, and hence the projection of history h onto channel c is a sequence $tr(h, c) = c(e_1) \dots c(e_n)$. We call such sequences *projected traces* and denote them by $TRproj$.

2.2 Agents and Processes

In multicast protocols, like the PGM, agents are defined to work independently of the identity and number of their upward and downward channels, because the architecture of the network is not known *a priori*. Our notion of *agents* intends to be very general, while respecting this limitation.

An *agent* is a pair $A = (\rho, f)$, where $\rho\colon TR \times M \to 2^{M{\uparrow} \times M{\downarrow}}$ is the *input/output relation*, and $f\colon TRproj \times Act \times Bool \to Bool$ is the *filter*. Let us explain

[3] Throughout the paper we identify $\perp$ and the set $\{\perp\}$. This should cause no confusion.

this definition. Intuitively, an agent A selects the events that can be executed as a function of the past history, and of the current input message i. After receiving i, the agent selects an event in two steps. First, it nondeterministically selects the messages o_1, o_2 to be sent upwards and downwards, respectively, as a function of the current trace tr of actions and the input message i. Formally, $(o_1, o_2) \in \rho(tr, i)$. In the second step, the agent determines the subset of downward channels through which the message o_2 is sent. The agent examines each channel $c \in Ch$, and decides whether to send o_2 through it or not depending on the action $a = (i, o_1, o_2)$, the projection $tr(h, c)$ of h on channel c, and on whether the input i came via the channel c or not. Formally, o_2 is sent through c if $f(tr(h, c), a, b) = \mathit{true}$, where b is true iff i arrived through channel c.

A *process* is a triple $P = (ch, Ch, A)$, where A is an agent, Ch is a set of channels, and ch is a channel that does not belong to Ch. The set of *transitions* of the process P is the subset of $H \times E \times H$ containing the triples $(h, e, h \cdot e)$ (also denoted by $h \xrightarrow{e} h \cdot e$), such that e is an event that can be selected by A when h is the past history of the process.

Example: We formalise the input/output relation ρ_n of the network element agent of the PGM protocol in our framework. (The filter can be formalised analogously.)

$$\begin{aligned}
\rho_n(tr, \mathrm{rdata}(nr, trl)) &= \{(\bot, \mathrm{rdata}(nr, trl))\} \\
\rho_n(tr, \mathrm{pnak}(nr)) &= \{(\mathrm{pnak}(nr), \mathrm{ncf}(nr))\} \\
\rho_n(tr, \mathrm{snak}(nr)) &= \{(\mathrm{snak}(nr), \bot)\} \\
\rho_n(tr, \mathrm{odata}(nr, trl)) &= \{(\bot, \mathrm{odata}(nr, trl))\} \\
\rho_n(tr, \mathrm{ncf}(nr)) &= \{(\bot, \bot)\} \\
\rho_n(tr, \bot) &= \begin{cases} \{(\mathrm{snak}(nr), \bot)\} & \text{if } nr \in \mathit{set_repair}(tr) \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}$$

Note that the value of *set_repair* is fully determined by tr; $nr \in \mathit{set\_repair}(tr)$ if and only if tr contains the action $(\mathrm{pnak}(nr), \mathrm{pnak}(nr), \mathrm{ncf}(nr))$ (which adds nr to *set_repair*) and no later occurrence of the actions $(\mathrm{ncf}(nr), \bot, \mathrm{ncf}(nr))$ or $(\mathrm{rdata}(nr), \bot, \mathrm{rdata}(nr))$ (which remove nr). So *set_repair* provides an abstract view on a trace sufficient to define ρ_n.

2.3 Tree Networks and Protocols

Loosely speaking, a tree network is a network of processes with a tree topology; every process is connected to its parent and children by bidirectional channels.

Syntax. A finite tree T is a set of nodes together with a partial order $\leq_T$ satisfying the usual tree condition: if $\mathbf{n}_1 \leq_T \mathbf{n}$ and $\mathbf{n}_2 \leq_T \mathbf{n}$, then $\mathbf{n}_1 \leq_T \mathbf{n}_2$ or $\mathbf{n}_2 \leq_T \mathbf{n}_1$. We denote the child relation by $\prec$ (i.e., $\mathbf{n} \prec \mathbf{n}'$ if $\mathbf{n} <_T \mathbf{n}'$ and there is no $\mathbf{n}''$ such that $\mathbf{n} <_T \mathbf{n}'' <_T \mathbf{n}'$). We write $p(n)$ for the *parent* of $\mathbf{n}$, i.e., for the unique $\mathbf{n}'$ such that $\mathbf{n}' \prec \mathbf{n}$ or for the symbol "_" if there is not such $\mathbf{n}'$. If $\mathbf{n} \prec \mathbf{n}'$, then we call the pair $[\mathbf{n}, \mathbf{n}']$ a *channel*. We define the sets of downward channels of $\mathbf{n}$ in T as $Ch(\mathbf{n}, T) = \{[\mathbf{n}, \mathbf{n}'] \mid \mathbf{n} \prec \mathbf{n}'\}$, and let $ch(\mathbf{n}, T) = [p(\mathbf{n}), \mathbf{n}]$. If $\mathbf{n}$ is

the root, then the channel $[\_, \mathbf{n}]$ is considered to connect to the environment. If no confusion is possible, we shorten $ch(\mathbf{n}, T)$ and $Ch(\mathbf{n}, T)$ to $ch(\mathbf{n})$ and $Ch(\mathbf{n})$, respectively. Throughout this paper, for simplicity we assume that the sender only uses its downward channels and receivers only use their upward channel.

A *tree network* is a pair (T, A), where T is a finite tree, and A is a mapping that associates to each node $\mathbf{n} \in N$ an agent $A(\mathbf{n})$. The mapping *Proc* associates to each node $\mathbf{n}$ the process $Proc(\mathbf{n}) = (ch(\mathbf{n}), Ch(\mathbf{n}), A(\mathbf{n}))$.

We call the root of the tree the *sender* of the network, and denote it by $\mathbf{s}$. Maximal nodes w.r.t. $\leq_T$ are called *receivers*. All other nodes are called *internal*.

A tree network (T, A) is *homogeneous* if $A(\mathbf{n}) = A(\mathbf{n}')$ for every two internal nodes $\mathbf{n}, \mathbf{n}'$, and $A(\mathbf{r}) = A(\mathbf{r}')$ for every two receivers $\mathbf{r}$ and $\mathbf{r}'$. A *protocol* P is a set of three agents A_s, A_n, A_r. A protocol P defines a family $\mathcal{N}(P)$ of homogeneous tree networks, namely the tree networks (T, A) satisfying $A(\mathbf{s}) = A_s$, $A(\mathbf{n}) = A_n$ for all internal elements $\mathbf{n}$, and $A(\mathbf{r}) = A_r$ for all receivers $\mathbf{r}$.

Semantics. A *network event* of a tree network (T, A) is a pair $(\mathbf{n}, e)$, where $\mathbf{n}$ is a node of T and e is an event of $Proc(\mathbf{n})$. Intuitively, a network event models that the process $Proc(\mathbf{n})$ executes the event e. We denote the set of all network events by *Nev*. A *network history* is a finite sequence of network events. Given a network history $nh = (\mathbf{n}_1, e_1) \dots (\mathbf{n}_n, e_n)$ and a node $\mathbf{n}$, we define by $nh(\mathbf{n})$ the projection of nh onto the events executed by $\mathbf{n}$. Notice that $nh(\mathbf{n})$ is a history of the process $Proc(\mathbf{n})$.

Since messages can overtake other messages, a channel behaves like a multiset, which only retains the multiplicity of each message, but not their order. Loss of a message need not be modelled explicitely, because it can be simulated by never taking the message from the channel. The multiset of messages that are waiting for delivery in the channel c after the execution of nh is denoted by $M(nh, c)$. Formally, $M(nh, [\mathbf{n}, \mathbf{n}'])$ is the multiset of messages sent through channel c by $Proc(\mathbf{n})$ and $Proc(\mathbf{n}')$ during the network history nh, minus the multiset of messages received through c by the same processes, also during nh.

A triple $(nh, (\mathbf{n}, e), nh')$, where nh, nh' are network histories and $(\mathbf{n}, e)$ is a network event, is a transition of (T, A) if there is a transition $h \xrightarrow{e} h'$ of $Proc(\mathbf{n})$ satisfying the following conditions:

(1) $nh(\mathbf{n}) = h$, $nh'(\mathbf{n}) = h'$, and $nh(\mathbf{n}') = nh(\mathbf{n})$ for every $\mathbf{n}' \neq \mathbf{n}$.
(2) Let i be the message received by $Proc(\mathbf{n})$ in e, and let c be the channel through which i arrived. Then, either $i = \perp$, or $i \in M(nh, c)$. I.e., the message received by $Proc(\mathbf{n})$ in the event e was either empty or it was waiting for delivery in the channel c.

If $(nh, (\mathbf{n}, e), nh')$ is a transition of (T, A), then we write $nh \xrightarrow[\mathbf{n}]{e} nh'$. We write $nh \xrightarrow[\mathbf{n}_1 \dots \mathbf{n}_n]{e_1 \dots e_n} nh'$ if there are transitions $nh \xrightarrow[\mathbf{n}_1]{e_1} nh_1 \dots nh_{n-1} \xrightarrow[\mathbf{n}_n]{e_n} nh'$. W.l.o.g, we assume that every network history has at least one successor (if not, just add self-looping transitions with some special label).

Fair executions. In our framework, a state of a tree network is given by a network history. An *execution* of T is an infinite sequence $\pi = nh_0\, nh_1\, nh_2 \ldots$ of network histories (i.e., of states) such that for every $i \geq 0$ there is a network event $(e_i, \mathbf{n}_i)$ satisfying $nh_i \xrightarrow[\mathbf{n}_i]{e_i} nh_{i+1}$. An execution is *fair* with respect to a subset $\tilde{T} \subseteq T$ if for every node $\mathbf{n} \in \tilde{T}$ in the network, $\mathbf{n}_i = \mathbf{n}$ for infinitely many $i \geq 0$, i.e., if every node in $\tilde{T}$ executes infinitely many events.

3 Fair Stuttering Simulations for Tree Networks

Fix a protocol P. We are interested in properties that concern only the behaviours of the sender and the receivers, since these are the 'visible' elements of the protocol. So we consider properties Φ of the form $\Phi = \forall\mu\forall\sigma\phi$, where μ is a tuple of *message variables*, σ is a tuple of *receiver variables*, and ϕ is a stuttering invariant LTL temporal logic formula [Lam83,PW97]. The atomic propositions of ϕ can be indexed by the variables of μ, σ.

Example: Informally, the main property that the PGM protocol should satisfy is "for every receiver $\mathbf{r}$ and for every message m sent by the sender, eventually one of the following two holds: $\mathbf{r}$ receives m, or $\mathbf{r}$ knows that m is lost, and that the sender is not going to resend it in the future". The second possibility means that $\mathbf{r}$ finds out that the lower end of the sender's retransmission window (given by *txw_trail*) is larger than the number of messages m.

Given a tree network $T \in \mathcal{N}(P)$, we interpret atomic propositions over sets of network histories of T.[4] We say that T satisfies Φ if for all valuations *val* of the variables μ, σ, and for all executions π of T that are fair with respect to $\mathbf{s}$ and to all receivers in the image of *val*, $\pi \models \phi[\mu := val(\mu),\ \sigma := val(\sigma)]$.

Note that executions in which some processes not mentioned in the property are not scheduled infinitely often are fair, and so the property must also hold for them. But we exclude as unfair those executions in which the sender or some receiver mentioned in the property is 'cut off from the network' after a certain time point, i.e. does no longer receive messages due, say, to a connection breakdown. We say that P satisfies Φ if T satisfies Φ for all $T \in \mathcal{N}(P)$, i.e., P satisfies a property if all the homogeneous tree networks of $\mathcal{N}(P)$ satisfy it.

A simulation of T by T' preserves a property Φ if $T' \models \Phi$ implies $T \models \Phi$. Our goal is to prove that all the tree networks of $\mathcal{N}(P)$ can be simulated by those in a subset $\mathcal{N}'(P) \subseteq \mathcal{N}(P)$, according to a notion of simulation that preserves properties. Once this is achieved, we can prove that P satisfies Φ by showing that the networks of $\mathcal{N}'(P)$ satisfy Φ.

We now define a stuttering version of simulation, similar to stuttering bisimulation [BCG88]. In the simulations used in our proofs all actions of the simulated network T have a corresponding sequence of actions in the simulating network T', and so we do not have to consider stuttering in T, which simplifies

[4] Once P is fixed, all networks (T, A) of $\mathcal{N}(P)$ share the same agent function A. So we shorten (T, A) to T.

the definition. Since we only require fair paths to satisfy a property, we adapt the definition accordingly, like in fair simulation as introduced in [GL94], the coarsest simulation that preserves fair-ACTL*.

Given two networks T and T', let Im be a mapping that assigns to each receiver $\mathbf{r}$ of T a receiver $Im(\mathbf{r})$ of T', called the *image* of $\mathbf{r}$. Let $Match(nh, nh')$ be the relation between histories of T and T' given by $tr(nh(\mathbf{s})) = tr(nh'(\mathbf{s}')))$, where $\mathbf{s}$ and $\mathbf{s}'$ are the senders of T and T', respectively, and $tr(nh(\mathbf{r})) = tr(nh'(Im(\mathbf{r})))$ for all receivers $\mathbf{r}$.

Definition 1
A *fair stuttering simulation of T by T' with respect to Im* is a relation $R \subseteq NH \times NH'$ such that $R(nh, nh')$ implies:

- $Match(nh, nh')$.
- For every subset $\tilde{T}$ of T consisting of the sender and receivers, and for every execution π of T starting at nh which is fair with respect to $\tilde{T}$ there is an execution π' of T' starting at nh', fair with respect to $Im[\tilde{T}]$, and an increasing mapping $\sigma\colon \mathbb{N} \longrightarrow \mathbb{N}$ such that (a) for all $n \geq 0$, $R(\pi(n), \pi'(\sigma(n))$, and (b) for all $\sigma(n) < j \leq \sigma(n+1)$, $Match(\pi(n+1), \pi'(j))$.

We say that T is simulated by T' (with respect to Im) if there is a mapping Im and a fair stuttering simulation R of T by T' with respect to Im such that $R(nh_0, nh'_0)$, where nh_0 and nh'_0 are the empty network histories of T and T'.

The following theorem describes properties that are preserved by fair stuttering simulation. It follows easily from the fact that for every execution π in T that is fair with respect to a subset $\tilde{T}$ there is an execution π' in T', fair with respect to $Im[\tilde{T}]$, such that the states of π and π' pointwise (up to stuttering) satisfy *Match*. This implies that π' satisfies the same stuttering-invariant LTL properties as π.

Theorem 1 *Let T and T' be tree networks such that T is simulated by T' with respect to Im. Let $\Phi = \forall\mu\forall\sigma\phi$ be a formula such that (1) ϕ is stuttering-invariant, and (2) for all atomic propositions p of ϕ and for all network histories nh and nh', $Match(nh, nh')$ implies $nh \models p \Longleftrightarrow nh' \models p$. Then, $T' \models \Phi$ implies $T \models \Phi$.*

Finally, we have the result we were looking for:

Theorem 2 *Let $\mathcal{N}' \subseteq \mathcal{N}(P)$ such that each $T \in \mathcal{N}(P)$ is simulated by some $T' \in \mathcal{N}'$, and let Φ as is Theorem 1. Then $\mathcal{N}' \models \Phi$ implies $\mathcal{N}(P) \models \Phi$.*

4 Collapsable Tree Networks

In this section, we explore conditions on protocols that allow to flatten the tree hierarchy, i.e. conditions implying that any tree network can be simulated by one with at most one layer of internal elements.

Throughout this section we fix a protocol P consisting of agents (ρ_s, f_s), (ρ_n, f_n) and (ρ_r, f_r). Let $\mathcal{N}^0(P)$ be the set of tree networks in $\mathcal{N}(P)$ without internal elements, and let $\mathcal{N}^1(P)$ be the set of tree networks in $\mathcal{N}(P)$ with only one layer of internal elements. For simplicity, we only consider sender agents that always sends downward messages through all their downward channels.

We are interested in protocols like the PGM, where the sender exchanges messages with the receivers, and the primary functionality of internal elements is to *forward* these messages. Transition $\mathbf{e_5}$ of the PGM provides an example. However, in order to deal with lost messages and to improve the efficiency, internal elements can also perform other tasks. First, they can *filter* downward messages: Instead of sending a message to all its successors, they select a subset of them as recipients. An example is transition $\mathbf{e_3}$. Moreover, besides forwarding with or without filtering, internal elements can also *generate* messages. Transition $\mathbf{e_6}$ is an example of 'spontaneous' generation, while in transition $\mathbf{e_1}$ reception of pnak(nr) triggers the generation of ncf(nr). The generation of messages can in general depend on the internal state of the component, i.e., its history. For example, generation of snak(nr) depends on whether nr is contained in the set *set_repair*.

We present our main result in two steps. First, in section 4.1 we consider internal agents that have only forwarding transitions (but possibly with filtering). We then consider protocols in which network elements can also generate messages. It is easy to see that for general protocols of this form there is no $n \geq 0$ such that $\mathcal{N}^n(P)$ simulates $\mathcal{N}(P)$. We identify a class of protocols P for which $\mathcal{N}(P)$ can be simulated by $\mathcal{N}^1(P)$, and show that the PGM belongs to it.

4.1 Forwarding Agents

Intuitively, a forwarding agent is an agent that forwards incoming upward messages through its upward channel, and multicasts incoming downward messages through some of its downward channels. We also allow the agent to 'swallow' messages.

Definition 2
Formally, an agent (ρ, f) is a *forwarding agent* if for every trace tr and every message $i \in M$:

- If $i \in M\uparrow$, then $\rho(tr, i) \subseteq \{(i, \bot), (\bot, \bot)\}$;
- if $i \in M\downarrow$, then $\rho(tr, i) \subseteq \{(\bot, i), (\bot, \bot)\}$; and
- if $i = \bot$, then $\rho(tr, \bot) = \{(\bot, \bot)\}$.

A *forwarding protocol* is a protocol whose network element agent (but not necessarily its sender or receiver agents) is forwarding.

Let T be a tree network. We define the tree $\underline{T}$ as follows: $\underline{T}$ contains a sender $\underline{\mathbf{s}}$ and a receiver $\underline{\mathbf{r}}$ for every receiver $\mathbf{r}$ of T, but no internal elements; $\leq_{\underline{T}}$ is the projection of $\leq_T$ onto $\underline{T}$. So in $\underline{T}$, every receiver is a child of the sender.

In the full version [EM02], we define a relation $nh \lhd \underline{nh}$ between network histories nh and $\underline{nh}$, and show that $\lhd$ is a fair stuttering simulation of T by $\underline{T}$. We get as corollary:

Theorem 3 *If P is a forwarding protocol and Φ is like in Theorem 1, then $\mathcal{N}^0(P) \models \Phi$ if and only if $\mathcal{N}(P) \models \Phi$.*

4.2 Forwarding and Generating Agents

In this section, we consider internal agents that can not only forward but also generate messages. Clearly, we can no longer expect $\mathcal{N}(P)$ to be simulated by $\mathcal{N}^0(P)$, unless the messages generated by the internal elements have no effects whatsoever. We give conditions under which $\mathcal{N}(P)$ can be simulated by $\mathcal{N}^1(P)$. We then show that these conditions are satisfied by the PGM protocol.

Let $M_{gen} \subseteq M$ be the set of messages that can be generated by internal network elements. (Notice that the sender and the receivers can also generate messages, but these do not have to be in M_{gen}.)

Definition 3
A *forwarding and generating agent* (*f&g agent* for short) is an agent (ρ, f) such that for every trace $tr \in TR$ and for every message $i \in M$, the following conditions hold:

- If $i \in M\uparrow$ and $(o_1, o_2) \in \rho(tr, i)$, then $o_1 \in \{i, \perp\}$ and $o_2 \in M_{gen} \cup \{\perp\}$;
- if $i \in M\downarrow$ and $(o_1, o_2) \in \rho(tr, i)$, then $o_1 \in M_{gen} \cup \{\perp\}$ and $o_2 \in \{i, \perp\}$; and
- if $i = \perp$ and $(o_1, o_2) \in \rho(tr, \perp)$, then $o_1, o_2 \in M_{gen} \cup \{\perp\}$.

A *f&g protocol* is a protocol with a f&g internal element agent.

These definition allows messages to be generated as 'side-effects' of forwarding other messages, or spontaneously, i.e. without receiving input. In the PGM protocol, ncf-messages are generated as side-effects, while snak-messages are generated spontaneously.

Conditions on the Protocol. We define the class of *simple protocols*, for which we prove that $\mathcal{N}^1(P)$ simulates $\mathcal{N}(P)$. This requires some preliminaries.

Receiving a message has two effects. The first, immediate effect is that some messages are sent. The second effect is that the internal state of the process (given in our model by the history) changes. This change may enable the process to send messages, but these may not be sent immediately. The change of state may also disable the emission of messages. An example of an enabling effect is given by transitions **e1** and **e6** of the PGM: Transition **e1** adds *nr* to *set_repair*, which enables the process to send snak through transition **e6**.

We need a definition implying the following intuitive idea: Receiving a message i in a subset M' may have arbitrary disabling effects, but it can only enable upward forwarding of i, or generation of messages in another subset M''.

Definition 4
Let (ρ, f) be an agent, tr be a trace of it, $M' \subseteq M$, and $M'' \subseteq M\uparrow$. Let $Rem(tr, M', M'')$ be the set of all traces resulting from tr by removing arbitrarily many actions of the form (i, o_1, o_2) such that either $i \in M'$ or $i = o_2 = \perp$ and $o_1 \in M''$. We say that M' *only enables* M'' in the agent (ρ, f) if for every trace tr and every $tr' \in Rem(tr, M', M'')$ the following conditions hold:

(1) If $i \in M' \cap M\uparrow$ and $(o_1, o_2) \in \rho(tr, i)$, then $o_1 \in M'' \cup \{i, \perp\}$ and $o_2 = \perp$;
if $i \in M' \cap M\downarrow$ and $(o_1, o_2) \in \rho(tr, i)$, then $o_1 \in M'' \cup \perp$ and $o_2 = \perp$;
(2) if $i \in M \setminus \perp$, then $\rho(tr, i) \subseteq \rho(tr', i)$; and
$\rho(tr, \perp) \setminus \rho(tr', \perp) \subseteq \{(o, \perp) \mid o \in M''\}$.

Let us see why this definition captures the intuition above. Condition (1) expresses that the *immediate* effect of receiving $i \in M'$ can only be forwarding i up, or the generation of an upward message $o_1 \in M''$. Condition (2) expresses that the *long-term* effect of receiving i can only the be generation of upward messages in M'', let us see why. Suppose first that $M'' = \emptyset$. Then (2) implies that receiving messages in M' can only have disabling effects: Whatever we can do after receiving the messages (given by $\rho(tr, i)$), we can also do without receiving them (given by $\rho(tr', i)$). Now consider the general case. Since $\rho(tr, \perp) \setminus \rho(tr', \perp) \subseteq \{(o, \perp) \mid o \in M''\}$, receiving messages in i may now enable actions $(\perp, o, \perp)$ for $o \in M''$. And since in $Rem(tr, M, M'')$ we now allow to remove actions $(\perp, o, \perp)$ for $o \in M''$, we make sure that such actions themselves only enable actions of the same kind.

We are almost ready to present the definition of simple protocols. Let M_{fil} be the subset of $M\downarrow$ containing the messages that can be filtered when being forward downwards. More precisely, for all actions a in which the downwards output message belongs to $M\downarrow \setminus M_{fil}$, there is no filtering, i.e., $f(tr, a, b) = true$ for all traces tr and all boolean values b. We now define:

Definition 5
Let $P = (A_s, A_n, A_r)$ be a f&g protocol. P is *simple* if:

(a) In A_n (the network element agent), M_{gen} only enables $\emptyset$, and
(b) in A_r (the receiver agent), $M_{fil} \cup M_{gen}$ only enables $M_{gen} \cap M\uparrow$.

If condition (a) is dropped, the following scenario becomes possible: A network element **n** spontaneously generates a message o_1 and sends it upwards to its parent. The parent forwards it up and, in the next step, generates itself another copy of o_1; all predecessors of **n** behave in the same way. This produces a cascade of upward messages, and the number of o_1's received by the sender depends on the position of **n** in the tree structure. It is easy to see that this makes it impossible to simulate arbitrary structures by structures with at most one layer of internal elements.

If condition (b) is dropped, the following scenario becomes possible: A downward message $m \in M_{fil}$ is filtered out by a network element **n**, i.e., **n** does not forward m down to any of its successors. The receivers that get m react by sending upwards a message $o_1 \notin M_{gen}$. Network elements that receive o_1 forward it

up and, in the next step, generate a message $o_1' \in M_{gen}$ and send it upwards. Again, the number of o_1' messages received by the sender depends on the position of $\mathbf{n}$ in the tree structure.

The simulation. We show that $\mathcal{N}^1(P)$ simulates $\mathcal{N}(P)$ if P is simple. Let T be a tree network. We define the tree $\underline{T}$ as follows: $\underline{T}$ contains a sender $\underline{\mathbf{s}}$. As network elements can generate messages, the simulating tree network now has to contain a network element $\underline{\mathbf{n}}$ for every network element $\mathbf{n}$ in T and $\underline{\mathbf{n}}$ is a child of $\underline{\mathbf{s}}$. Moreover, a network element $\underline{\mathbf{n}}$ has to have the same receivers below itself as $\mathbf{n}$ does, because the actions of all these receivers can affect $\mathbf{n}$. So for every $\mathbf{r} > \mathbf{n}$, $\underline{\mathbf{n}}$ needs to have a child $\underline{\mathbf{r}}(\mathbf{n})$ that acts like $\mathbf{r}$. Thereby, we assume for simplicity that all receivers have an internal element as parent.

Figure 1 displays an example tree T and its flattening $\underline{T}$. In the full version [EM02], we define a simulation relation $nh \lhd \underline{nh}$ between network histories nh of T and $\underline{nh}$ of $\underline{T}$. The receiver $\underline{\mathbf{r}}(p(\mathbf{r}))$ exactly simulates $\underline{\mathbf{r}}$, and so we abbreviate $\underline{\mathbf{r}}(p(\mathbf{r}))$ by $\underline{\mathbf{r}}$. The other copies $\underline{\mathbf{r}}(\mathbf{n})$ are guaranteed to be able to execute the same actions as $\mathbf{r}$ except actions (i, o_1, o_2) such that $i \in M_{fil} \cup M_{gen}$ or actions $(\bot, o, \bot)$ such that $o \in M_{gen}$.

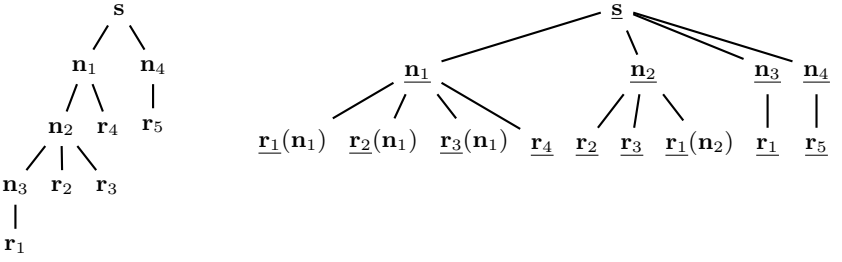

Fig. 1. Example of a tree T and its flattening $\underline{T}$

We obtain that for simple f&g protocols, it suffices to consider trees with only one level of internal elements:

Theorem 4 *If P is a simple f&g protocol and Φ is as in Theorem 1 then $\mathcal{N}^1(P) \models \Phi$ holds if and only if $\mathcal{N}(P) \models \Phi$.*

The PGM protocol. We sketch why the PGM example as presented in Section 1 is simple in the sense of Definition 5. In the PGM protocol, M_{gen} consists of the messages of form snak(nr) and ncf(nr), and M_{fil} consists of the messages of form rdata(nr, trl). Let us show that the protocol satisfies the conditions of Definition 5.

First, the network element agent (ρ_n, f_n) of page 133 is f&g by definition. For (a): When forwarding snak(nr), only a message in M_{gen} (namely snak(nr)) is sent upwards, and nothing downwards, and on reception of ncf(nr) nothing is sent. This shows that condition (1) of Definition 4 holds. In order to see that

messages in M_{gen} do not have long-term enabling effects (condition (2)), first note that ρ_n can be defined in terms of the variable *set_repair*, as explained on 133.[5] Reception of snak(nr) does not change *set_repair*, and on reception of ncf(nr), nr is removed from *set_repair*: So if $tr' \in Rem(tr, M_{gen}, \emptyset)$, i.e., if tr' is obtained by removing actions with input snak(nr) or ncf(nr) then in tr' the same actions as in tr (and possibly more) are enabled. For (b): On reception of rdata(nr,trl) or ncf(nr), nothing is sent (condition (1)). In order to see that messages in M_{gen} do not have long-term enabling effects, notice first that ρ_n can be defined in terms of the variables of the agent. Since reception of rdata(nr,trl) adds nr to *set_nr* and removes it from *set_missing* and *set_smissing*, it only disables actions. Reception of ncf(nr) only enables actions of the form $(\bot, \text{snak}(nr'), \bot)$. Finally, sending of $(\bot, \text{snak}(nr'), \bot)$ does not change the variables of the receiver, and so it does not enable or disable actions. So the only actions that can be enabled in tr but not in $tr' \in Rem(tr, M_{fil} \cup M_{gen}, M_{gen} \cap M\uparrow)$ are of the form $(\bot, \text{snak}(nr'), \bot)$.

5 Reduction of One-Layer Tree Networks

Given a simple protocol P and a property Φ, we have shown that checking Φ reduces to proving that $\mathcal{N}^1(P) \models \Phi$. We now introduce the class of *iteration protocols*, and show that for them $\mathcal{N}^1(P) \models \Phi$ reduces to proving $U_k \models \Phi$, where U_k is a particular instantiation of $\mathcal{N}^1(P)$ that depends on the number k of receiver variables used in Φ. Combining the two results we have that a simple iteration protocol P satisfies Φ if and only if $U_k \models \Phi$.

A trace tr is *reachable* by an agent (ρ, f) if it is empty or if $tr = tr' \cdot (i, o_1, o_2)$ where tr' is reachable and $(o_1, o_2) \in \rho(tr', i)$. An agent *can resend upward messages* if for every reachable trace $tr \cdot (i, o_1, o_2)$ and for every $n \geq 1$, the trace $tr \cdot (i, o_1, o_2) \cdot (\bot, o_1, \bot)^n$ is also reachable.

Definition 6
A protocol is an *iteration protocol* if $M\uparrow$ is a finite set, and both the receiver and the network element agents can resend upward messages.

If we bound the number of possible message numbers in the PGM protocol, i.e., instead of $nr \in \mathbb{N}$ we say $nr \in [1..n]$ for some number n, then we obtain a simple iteration protocol: The conditions of Definition 6 hold because of transitions **e6**, **r4** and **r5**.

Definition 7
Let P be an iteration protocol P, and let u and g be the sizes of $M\uparrow$ and $M\uparrow \cap M_{gen}$, respectively. For each $n \geq 1$, we define the *universal instance* U_k of $\mathcal{N}^1(P)$ as follows: The sender has $k + g + u$ children, all of them network elements; each of the first $k+u$ network elements has $u+1$ children; and each of the other g network elements has u children. For $1 \leq i \leq k$, we denote the first child of the i-th network element by $\underline{\mathbf{r}}_i$. We also denote the tuple $(\underline{\mathbf{r}}_1, \ldots, \underline{\mathbf{r}}_k)$ by $\underline{\mathbf{R}}$.

[5] The variable *set_interf* is irrelevant for ρ_n, it only affects the filter function f_n.

In order to prove that U_k can simulate the behaviour of instances T in $\mathcal{N}^1(P)$, we use the following notion: Let Im be a function mapping a k-sized subset $\tilde{R} = \{\mathbf{r}_1, \ldots, \mathbf{r}_k\}$ of receivers of T to receivers of U_k. Let $Match(nh, nh')$ hold if $tr(nh)(\mathbf{r}) = trl(nh'(Im(\mathbf{r})))$ for all $\mathbf{r} \in \tilde{R}$. We say that fair executions of T are *stuttering-included* in the fair executions of U_k if for any execution π which is fair with respect to $\tilde{R}$, there is an execution π' of U_k, fair with respect to $Im[\tilde{R}]$, and an increasing mapping $\sigma : \mathbb{N} \to \mathbb{N}$ such that $Match(\pi(i), \pi'(\sigma(i)))$ holds for all i, and for all $\sigma(i) < j \leq \sigma(i+1)$, $Match(\pi(i+1), \pi'(j))$.

Intuitively, in an iteration protocol, u-many receivers can mimic the behaviour of any set R of receivers as follows: For every $m \in M\uparrow$ there is a receiver $\mathbf{r}(m) \in R$ that first outputs m. We simulate all transitions of $\mathbf{r}(m)$ until the first output of m, and switch to iterating $(\perp, m, \perp)$ afterwards, and so can simulate all actions of receivers in R. By using this observation, we obtain:

Theorem 5 *Let P be a simple f&g iteration protocol and let $T \in \mathcal{N}^1(P)$. Let $\{\mathbf{r}_1, \ldots, \mathbf{r}_k\}$ be a subset of the receivers of T. The fair executions of T are stuttering-included in the fair executions of U_k with respect to the mapping Im given by $Im(\mathbf{r}_i) = \underline{\mathbf{r}}_i$.*

Analogously to Theorem 1, stuttering-inclusion of fair executions implies that any formula Φ with k receiver variables that holds for U_k also holds for T. As U_k is in $\mathcal{N}^1(P)$, we obtain:

Theorem 6 *Let P be a simple f&g iteration protocol, and let Φ be a property $\Phi = \forall\mu\forall\sigma\phi$ as in Theorem 1 such that σ is a tuple of n receiver variables. Then, $\mathcal{N}^1(P) \models \Phi$ if and only if $U_n \models \Phi[\sigma := \underline{\mathbf{R}}]$ and so, by Theorem 4, P satisfies Φ if and only if $U_n \models \Phi[\sigma := \underline{\mathbf{R}}]$.*

6 Conclusions and Related Work

We have provided a general formal model of multicast network protocols with which tree-based multicast protocols can be modelled appropriately. We have proved a general theorem showing that for a simple class of protocols, the verification problem reduces to the analysis of instantiations with at most one layer of internal elements between sender and receivers. For a smaller class we have also proved that the verification reduces to the analysis of one single instantiation. Protocols whose internal elements just forward messages fit easily in our class. In fact, we have shown that the PGM protocol, whose internal elements exhibit a far more complicated behaviour, also fits in it.

As future work, we plan to explore whether our results can also be used for protocols that use local error recovery, which is a possible extension of the PGM protocol, and whether our approach can be extended to the analysis of timed protocols.

Related work. Some work on regular model-checking has addressed the problem of automatically verifying systems with a parameterised tree structure

[BT02]. However, these techniques still seem far from being able to attack systems of the complexity of the PGM. There are also some papers on the analysis of the PGM protocol. However, so far they have concentrated on analysing the behaviour of a fixed instance, and so this work has a different nature to the work carried out here. In [BBP02], the timed behaviour of a small instance of a simplified model is analysed. In the ADVANCE project, the untimed behaviour of a system that can simulate the universal instance U_1 has been studied. By our results this system can simulate *any* instance with respect to the main property of the protocol, since this property only involves one receiver variable. Unfortunately, at the time of writing this paper this instance is still out of the reach of the automatic tools.

Acknowledgements. This work has been supported by the FP5 Project ADVANCE, contract No IST-1999-29082.

References

[BBP02] Bérard, B., Bouyer, P. and Petit, A. *Analysing the PGM protocol with UPPAAL*. In: *2nd Workshop on Real-Time Tools*. Dep. Information Technology, Uppsala Univ., 2002, Tech. Report 2002-025.

[BCG88] Browne, M. C., Clarke, E. and Grumberg, O. *Characterizing finite Kripke structures in propositional temporal logic*. Theoretical Computer Science, 59: 115–131, 1988.

[BT02] Bouajjani, A. and Touili, T. *Extrapolating tree transformations*. In: *Proc. 14th Intl. Conf. on Computer Aided Verification*. 2002, LNCS 2404.

[EM02] Esparza, J. and Maidl, M. *Simple representative instantiations for multicast protocols*, 2002. Available at `http://www.dcs.ed.ac.uk/monika`.

[EN95] Emerson, E. A. and Namjoshi, K. S. *Reasoning about rings*. In: *Proc. 22th ACM Conf. on Principles of Programming Languages*. 1995.

[GL94] Grumberg, O. and Long, D. E. *Model checking and modular verification*. TOPLAS, 16(3): 843–871, 1994.

[Lam83] Lamport, L. *What good is temporal logic?* In: *Proc. IFIP 9th World Computer Congress*. 1983.

[Mai02] Maidl, M. *Simple representative instantiations for the PGM protocol*, 2002. Available at `http://www.dcs.ed.ac.uk/monika`.

[PPV] Papadopoulos, C., Parulkar, G. and Varghese, G. *LMS: A router assisted scheme for reliable multicast*. To appear in: IEEE/ACM Transactions on Networking.

[PW97] Peled, D. and Wilke, T. *Stutter-invariant temporal properties are expressible without the next-operator*. Information Processing Letters, 63: 243–246, 1997.

[S+00] Speakman, T. et al. *PGM reliable transport protocol specification*, 2000. RFC 3208 (experimental) of the IETF. Available at: `http://www.ietf.org/rfc.html`.