Efficient Verification of a Multicast Protocol for Mobile Computing [1]

Giuseppe Anastasi, Alberto Bartoli*, Nicoletta De Francesco, Antonella Santone

Dipartimento di Ingegneria dell'Informazione, Università di Pisa, Italy
{anastasi,nico,santone}@iet.unipi.it

* Dipartimento di Elettrotecnica, Elettronica ed Informatica, Università di Trieste, Italy
{bartolia}@univ.trieste.it

Contact address:
Antonella Santone
Dipartimento di Ingegneria della Informazione
Università di Pisa
Via Diotisalvi, 2
I-56126 Pisa, Italy
fax: +39 050 568 522

Abstract. We present the formal verification of a multicast protocol for mobile computing. The protocol supports reliable and totally ordered communication within a set of processes running on mobile hosts. Mobile hosts communicate with a wired infrastructure through wireless links. The protocol is specified in CCS and checked using the Concurrency Workbench tool. The protocol was chosen as a case study to evaluate the usefulness of a methodology, by means of which a property is checked on a reduced system, where the reduction is driven by the formula expressing the property itself. The reduction is obtained by transforming the program into one having a smaller representation. The approach is based on a logic, the selective mu-calculus, which has the characteristic that each formula allows immediately pointing out the parts of the system that do not alter the truth value of the formula itself, and thus can be ignored. We show and discuss the experimental results obtained.

1 Introduction

The motivation for using Formal Description Techniques in the design of systems is to have an unambiguous description of them about which it is possible to reason. Being based on a formal framework, verification and testing of the specification can be performed in an automated way. Verification of a system consists in checking whether the system satisfies a set of required properties. When designing concurrent systems, "safety" and "liveness" properties are considered, which concern eventuality or necessity constraints on the occurrence of the events of the system. Temporal logic has been defined to express this kind of properties: it extends classical logic with eventuality and necessity operators.

The Calculus of Communicating Systems (CCS) [25] has been defined to be used in the specification of concurrent and distributed systems, since it contains synchronization, communication and parallelism as primitive notions. It is based on an algebraic theory of concurrency, allowing to express equivalence notions and system properties. The Concurrency Workbench of North Carolina (CWB-NC) [16, 18] is a verification environment including several different specification languages, among which CCS. The specifications can be checked for different equivalences, and different logics can be used to express properties. In the CWB-NC the verification of temporal logic formulae is based on *model checking* [14]: the formula ϕ that the system must satisfy is checked on a finite structure representing the system. This structure is a labeled transition system, i.e. an automaton whose transitions are labeled by event names.

The efficiency of model checking is essentially influenced by the number of states of the transition system. A problem of model checking is state explosion: concurrent systems are often represented by automata

[1] Work partially supported by "Progetto Giovani Ricercatori".

with a prohibitive number of states. On the other hand, often the property one wants to check does not concern the whole transition system, but only some parts of it. An approach to reduce the number of states is the definition of suitable abstraction criteria by means of which a smaller transition system can be obtained, including only the parts that "influence" the property. In [6] a temporal logic is proposed, called *selective mu-calculus*, which has the characteristic that each formula allows immediately pointing out the parts of the transition system that do not alter the truth value of the formula itself. In particular, given a logic formula ϕ of the selective mu-calculus, only the transitions labeled by the actions syntactically occurring in ϕ have to be considered. In [7,8] a methodology is proposed, based on the selective mu-calculus, by means of which, given a formula, the CCS process is syntactically transformed into a smaller one (corresponding to a reduced transition system), where the reduction is driven by the formula to be checked. The formula is then checked on the reduced transition system. A prototype tool has been defined, both for CCS [7] and for LOTOS [8], implementing the methodology. The prototype produces the reduced transition system in the CWB-NC format. The CWB-NC tool can then be used to check selective mu-calculus formulae, which can be easily translated into the temporal logic mu-calculus, available in the CWB-NC.

In the present paper we apply our methodology to a case study, i.e. a multicast protocol for mobile computing, developed by two of the authors [2,9]. The protocol supports reliable and totally ordered communication within a set of processes running on mobile hosts. Mobile hosts communicate with a wired infrastructure through wireless links. The protocol manages dynamic membership and exhibits some novel features. In particular, unlike similar protocols (e.g. [1]) it is not based on hand-off; it allows user movements at any time; and it is able to work correctly even with incomplete coverage of the wireless network. Simulation analysis has shown that the protocol is indeed practical as it exhibits good performance and scalability [3]. In the paper we specify the protocol in CCS and the desirable properties in selective mu-calculus and apply our reducing method to check the properties. We show and discuss the experimental results obtained.

2 Calculus of Communicating Systems

Let us now briefly recall the Calculus of Communicating Systems (CCS) [25]. The syntax of *processes* is the following:

$$p ::= nil \mid \alpha.p \mid p+p \mid \ p|p \ \mid p\backslash L \mid p[f] \mid x$$

where α ranges over a finite set of actions $\mathcal{A} = \{\tau, a, \overline{a}, b, \overline{b}, ...\}$. Input actions are labeled with "non-barred" names, e.g. a, while output actions are "barred", e.g. $\overline{a}$. The action $\tau \in \mathcal{A}$ is called *internal action*. The set L ranges over sets of *visible actions* $(\mathcal{A} - \{\tau\})$, f ranges over functions from actions to actions, while x ranges over a set of *constant* names: each constant x is defined by a constant definition $x \overset{def}{=} p$.

The process nil can perform no actions. The process $\alpha.p$ can perform the action α and thereby become the process p. The process $p+q$ can behave either as p or as q. The operator $|$ expresses parallel composition: if the process p can perform α and become p', then $p|q$ can perform α and become $p'|q$, and similarly for q. Furthermore, if p can perform a visible action l and become p', and q can perform $\overline{l}$ and become q', then $p|q$ can perform τ and become $p'|q'$. The operator $\backslash$ expresses the restriction of actions. If p can perform α and become p', then $p\backslash L$ can perform α to become $p'\backslash L$ only if $\alpha, \overline{\alpha} \notin L$. The operator $[f]$ expresses the relabeling of actions. If p can perform α and become p', then $p[f]$ can perform $f(\alpha)$ and become $p'[f]$. Each relabeling function f has the property that $f(\tau) = \tau$. Finally, a constant x behaves as p if $x \overset{def}{=} p$.

The operational semantics of a process p is a labeled transition system, i.e. an automaton whose states correspond to processes (the initial state corresponds to p) and whose transitions (arcs) are labeled by actions in $\mathcal{A}$. This automaton is called *standard transition system* of p and denoted by $S(p)$. The semantics of a process p is precisely defined by means of a structural operational semantics, see Table 1.

From now on, we shall consider only finite CCS processes, i.e. processes with finite standard transition systems. A sufficient condition for finiteness is that the parallel operator does not occur inside recursive process definitions.

Act	$\dfrac{}{\alpha.p \xrightarrow{\alpha} p}$	**Sum**	$\dfrac{p \xrightarrow{\alpha} p'}{p+q \xrightarrow{\alpha} p'}$	**Con**	$\dfrac{p \xrightarrow{\alpha} p'}{x \xrightarrow{\alpha} p'}\ x \stackrel{def}{=} p$
Par	$\dfrac{p \xrightarrow{\alpha} p'}{p \vert q \xrightarrow{\alpha} p' \vert q}$	**Com**	$\dfrac{p \xrightarrow{l} p',\ q \xrightarrow{\overline{l}} q'}{p \vert q \xrightarrow{\tau} p' \vert q'}$	**Rel**	$\dfrac{p \xrightarrow{\alpha} p'}{p[f] \xrightarrow{f(\alpha)} p'[f]}$
		Res	$\dfrac{p \xrightarrow{\alpha} p'}{p\backslash L \xrightarrow{\alpha} p'\backslash L}\ \alpha, \overline{\alpha} \notin L$		

Table 1. *Operational semantics of CCS*

3 Model checking and selective mu-calculus

In the model checking framework, systems are modeled as transition systems and requirements are expressed as formulae in temporal logic. A model checker then accepts two inputs, a transition system and a temporal formula, and returns "true" if the system satisfies the formula and "false" otherwise. The logic *selective mu-calculus* [5, 6] was defined with the goal of reducing the number of states of the transition systems in such a way that the reduction is driven by the formulae to be checked, and in particular by the syntactic structure of the formulae. The selective mu-calculus is a variant of mu-calculus [24, 28], and differs from it in the definition of the modal operators. The syntax of the selective mu-calculus is the following, where K and R range over sets of actions, while Z ranges over a set of variables:

$$\phi ::= \mathtt{tt} \mid \mathtt{ff} \mid Z \mid \phi_1 \vee \phi_2 \mid \phi_1 \wedge \phi_2 \mid\ [K]_R\, \phi \mid\ \langle K \rangle_R\, \phi \mid \nu Z.\phi \mid \mu Z.\phi$$

The satisfaction of a formula ϕ by a state s of a transition system, written $s \models \phi$, is defined as follows: each state satisfies $\mathtt{tt}$ and no state satisfies $\mathtt{ff}$; a state satisfies $\phi_1 \vee \phi_2$ ($\phi_1 \wedge \phi_2$) if it satisfies ϕ_1 or (and) ϕ_2. $[K]_R\, \phi$ and $\langle K \rangle_R\, \phi$ are the selective modal operators: they require that the formula ϕ is satisfied after the execution of an action of K, provided that it is not preceded by any action in $K \cup R$. More precisely:

$[K]_R\, \phi$ is satisfied by a state which, for every performance of a sequence of actions not belonging to $R \cup K$, followed by an action in K, evolves in a state obeying ϕ.

$\langle K \rangle_R\, \phi$ is satisfied by a state which can evolve to a state obeying ϕ by performing a sequence of actions not belonging to $R \cup K$ followed by an action in K.

The precise definition of the satisfaction of a closed formula φ by a state is given in Table 2.

As in standard mu-calculus, a fixed point formula has the form $\mu Z.\phi$ ($\nu Z.\phi$) where μZ (νZ) *binds* free occurrences of Z in ϕ. An occurrence of Z is free if it is not within the scope of a binder μZ (νZ). A formula is *closed* if it contains no free variables. $\mu Z.\phi$ is the least fix-point of the recursive equation $Z = \phi$, while $\nu Z.\phi$ is the greatest one. To give an intuition of their meaning, consider the formulae $\phi = \mu Z.(\psi \vee \langle a \rangle_\emptyset\, Z)$ and $\phi' = \nu Z.(\psi \wedge [a]_\emptyset\, Z)$. A transition system satisfies ϕ if it can evolve to a state satisfying ψ after a finite number of occurrences of action a (ignoring all other actions), while it satisfies ϕ' if it satisfies ψ along any path containing a (ignoring all other actions).

A transition system T satisfies a formula ϕ, written $T \models \phi$, if and only if $q \models \phi$, where q is the initial state of T. A CCS process p satisfies ϕ if $S(p) \models \phi$.

The selective mu-calculus is equivalent to the mu-calculus. In fact it is easy to see that the standard mu-calculus modal operators $[K]\phi$ and $\langle K \rangle \phi$ can be defined by means of the selective operators subscribed by the whole set of actions $\mathcal{A}$:

$$[K]\phi = [K]_{\mathcal{A}}\, \phi \text{ and } \langle K \rangle \phi = \langle K \rangle_{\mathcal{A}}\, \phi$$

$$
\begin{array}{lll}
p \not\models \texttt{ff} & & \\
p \models \texttt{tt} & & \\
p \models \varphi \wedge \psi & \text{iff} & p \models \varphi \text{ and } p \models \psi \\
p \models \varphi \vee \psi & \text{iff} & p \models \varphi \text{ or } p \models \psi \\
p \models [K]_R\, \varphi & \text{iff} & \forall p'.\forall \alpha \in K. p \overset{\alpha}{\Longrightarrow}_{K \cup R} p' \text{ implies } p' \models \varphi \\
p \models \langle K \rangle_R\, \varphi & \text{iff} & \exists p'.\exists \alpha \in K. p \overset{\alpha}{\Longrightarrow}_{K \cup R} p' \text{ and } p' \models \varphi \\
p \models \nu Z.\varphi & \text{iff} & p \models \nu Z^n.\varphi \text{ for all } n \\
p \models \mu Z.\varphi & \text{iff} & p \models \mu Z^n.\varphi \text{ for some } n
\end{array}
$$

where:

- for each n, $\nu Z^n.\varphi$ and $\mu Z^n.\varphi$ are defined as:

$$
\begin{array}{ll}
\nu Z^0.\varphi = \texttt{tt} & \mu Z^0.\varphi = \texttt{ff} \\
\nu Z^{n+1}.\varphi = \varphi[\nu Z^n.\varphi/Z] & \mu Z^{n+1}.\varphi = \varphi[\mu Z^n.\varphi/Z]
\end{array}
$$

  where the notation $\varphi[\psi/Z]$ indicates the substitution of ψ for every free occurrence of the variable Z in φ.
- $p \overset{\alpha}{\Longrightarrow}_I q$ iff $p \xrightarrow{\delta\alpha} q$, where $\delta \in (\mathcal{A} - I)^*$ and $I \subseteq \mathcal{A}$.

Table 2. Satisfaction of a closed formula by a state

On the other hand the selective operators can be expressed in standard mu-calculus as follows:

$$\langle K \rangle_R\, \phi \overset{def}{=} \mu Z.\langle K \rangle \phi \vee \langle \mathcal{A} - (K \cup R) \rangle Z$$
$$[K]_R\, \phi \overset{def}{=} \nu Z.[K]\phi \wedge [\mathcal{A} - (K \cup R)]Z$$

In the sequel we will use the following abbreviations (where K ranges over sets of actions and $\mathcal{A}$ is the set of all actions):

$$[\alpha_1, \dots, \alpha_n]_R\phi \overset{def}{=} [\{\alpha_1, \dots, \alpha_n\}]_R\phi$$
$$[-]_R\phi \overset{def}{=} [\mathcal{A}]_R\phi$$
$$[-K]_R\phi \overset{def}{=} [\mathcal{A} - K]_R\phi$$

Example 1. We give some examples of selective mu-calculus formulae to explain the use of the selective operators.

$\psi_1 = [\mathtt{a}]_{\{\mathtt{b}\}}\mathtt{ff}$: *"it is not possible to perform an action* a *if an action* b *has not been previously performed".*

$\psi_2 = \langle \mathtt{a} \rangle_\emptyset \mathtt{tt}$: *"it is possible to perform an action* a *preceded by any action".*

$\psi_3 = \nu \mathtt{Z}.[\mathtt{a}]_\emptyset(\mathtt{Z} \wedge [\mathtt{a}]_{\{\mathtt{b},\mathtt{c}\}}\mathtt{ff})$: *"it always holds that, after an action* a *has occurred, a successive* a *cannot occur before either an action* b *or an action* c *has occurred".*

Let us consider the transition systems in Figure 1. It holds that:

$$
\begin{array}{lll}
S1 \not\models \psi_1 & S2 \models \psi_1 & S3 \not\models \psi_1 \\
S1 \models \psi_2 & S2 \models \psi_2 & S3 \models \psi_2 \\
S1 \models \psi_3 & S2 \not\models \psi_3 & S3 \not\models \psi_3
\end{array}
$$

The basic characteristic of the selective mu-calculus is that the actions relevant for checking a formula ϕ are those ones explicitly mentioned in the modal operators used in the formula itself. Thus we define

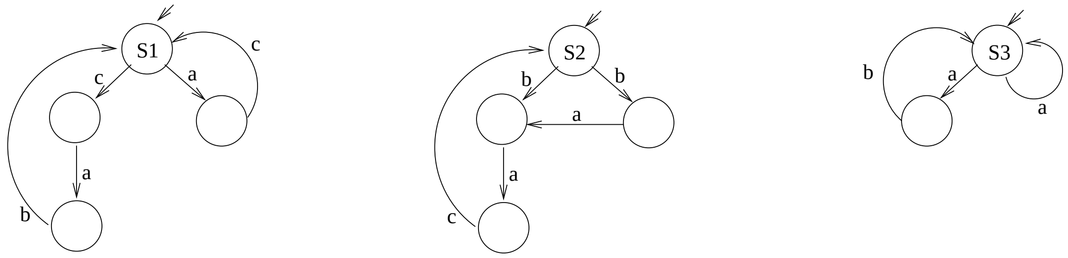

Fig. 1. Three transition systems

the set $\mathcal{O}(\phi)$ of *occurring actions* of a formula ϕ as the union of all sets K and R appearing in the modal operators ($[K]_R\psi$, $\langle K\rangle_R\psi$) occurring in ϕ.

In the works [5] and [6] ρ-equivalence is defined, formally characterizing the notion of "the same behavior with respect to a set ρ of actions":

*two transition systems are ρ-***equivalent** *if a ρ-bisimulation relating their initial states exists.*

The definition of ρ-bisimulation is based on the concept of α-*ending path*: an α-ending path is a sequence of transitions labeled by actions not in ρ followed by a transition labeled by the action α in ρ. Two states S_1 and S_2 are ρ-bisimilar if and only if for each α-ending path starting from S_1 and ending into S_1', there exists an α-ending path starting from S_2 and ending into a state ρ-bisimilar to S_1', and vice-versa.

In [6] it is proved that

two transition systems are ρ-equivalent if and only if they satisfy the same set of formulae with occurring actions in ρ.

As a consequence, a formula of the selective mu-calculus with occurring actions in ρ can be checked on any transition system ρ-equivalent to the standard one. Thus, improvements in model checking can be obtained by minimizing the transition system with respect to the actions in ρ. Obviously, the degree of reduction depends on the size of ρ with respect to the size of the whole set A of actions.

Example 2. Consider the transition systems illustrated in Figure 1. $S1$ is $\{a, b\}$-equivalent to $S3$. The two transition systems give the same value for the formulae containing only actions in $\{a, b\}$. In particular, they satisfy ψ_2, while they do not satisfy ψ_1. Note that $\mathcal{O}(\psi_1) = \{a, b\}$ and $\mathcal{O}(\psi_2) = \{a\} \subseteq \{a, b\}$. On the contrary $S2$ is not $\{a, b\}$-equivalent to $S3$, since it can perform an action b without performing an action a.

In [7] a method is defined, which, given a CCS process p and a set of interesting actions ρ occurring in a formula ϕ to be checked, transforms p into another process q, corresponding to a smaller transition system than that of p, on which ϕ can be equivalently checked. The tool is based on a set of syntactic transformations rules, having the form $p \longmapsto q$, where the application of the rule has the effect of rewriting p as q. The rules are defined in Table 3.

They consist in an *action deleting rule* and two *compacting rules*. The first one deletes (when possible) actions not belonging to ρ from a process, provided that they are not the first action of a choice branch in the scope of a parallel composition (this is necessary to preserve ρ-equivalence). If we apply this rule repeatedly, we can delete sequences of actions not in ρ. The other two rules compact processes by eliminating redundant branches of choices.

In [7], a transformation algorithm has been defined which, starting from a set ρ of actions and a process p, transforms p into a reduced process by applying the transformation rules until not possible. Note that, given an initial set of actions that cannot be deleted, when managing different sub-processes of a process p, we must consider possibly different sets of actions that can be deleted. Consider, for example, $\rho = \{a\}$ and the process

$$(a.b.a.nil \mid a.nil)\backslash\{b\}$$

Action deleting rule

$\alpha.p \longmapsto p$, if $\alpha \notin \rho$ and α is not the first action of a choice branch in the scope of a parallel composition

Compacting rules

$$C_1 : \alpha.p + p \quad \longmapsto \alpha.p \;\text{ if } \alpha \notin \rho$$
$$C_2 : \alpha.p + \beta.p \longmapsto \alpha.p \;\text{ if } \alpha, \beta \notin \rho$$

Table 3. The transformation rules

which performs the sequence $(a\ a)$ and then stops, since the communication on b cannot occur (the second operand does not perform b). If instead we delete b from the first operand, a third a can be performed. Thus we cannot delete any action occurring in a parallel context on which a synchronization can occur, without altering ρ-equivalence. In order to take into account this fact, the transformation algorithm transforms the sub-processes p_1 and p_2 of $(p_1 \mid p_2)\backslash L$ starting from the set of interesting actions $\rho \cup L$. Similarly, when deleting actions not in ρ from a relabeled process, we must consider as set of interesting actions the set $f^{-1}(\rho) = \{\alpha \mid f(\alpha) \in \rho\}$, since now, the interesting actions are also those ones relabeled by f into actions in ρ.

Example 3. For example consider the process:

$$p = (a.d.b.nil \mid \overline{a}.c.nil)\backslash\{a\}[f] \textit{ where } f(b) = c,\ f(c) = b$$

If we apply the transformation algorithm with $\rho = \{c\}$ we obtain:

$$p \longmapsto (a.b.nil \mid \overline{a}.nil)\backslash\{a\}[f]$$

In fact:

$a.d.b.nil \longmapsto a.b.nil$
$\quad\quad\{$*Action deleting rule with* $\rho = f^{-1}(\{c\} \cup \{a\}) = \{a, b\}\}$

and

$\overline{a}.c.nil \longmapsto \overline{a}.nil$
$\quad\quad\{$*Action deleting rule with* $\rho = f^{-1}(\{c\} \cup \{a\}) = \{a, b\}\}$

A prototype tool which implements the algorithm has been defined in [7]; it is written in SICStus Prolog [27]; the output is in the format of the CWB-NC tool: a file .ccs, containing the reduced CCS process. Thus, both the construction of the transition system and the checking of the properties can be made using the CWB-NC environment.

4 Protocol Overview

We shall refer to a simplified version of a *multicast* protocol for distributed *mobile* systems that provides *reliable* and *totally-ordered* communication. The full version of the protocol can be found in [9], along with a detailed discussion of its motivation and advantages. For brevity, we shall not mention explicitly the aspects of the original protocol that we have simplified. The protocol has been later extended to support weaker ordering guarantees that can be selected by the sender of each multicast, i.e., FIFO and causally ordered delivery [2].

We consider a system composed of mobile hosts (*MH*) and stationary hosts (*SH*) as shown in Figure 2.

Communication occurs solely via message-passing. SHs are connected to a wired network that provides reliable and FIFO-ordered communication. Some SHs, called *gateways*, are connected to the wired network

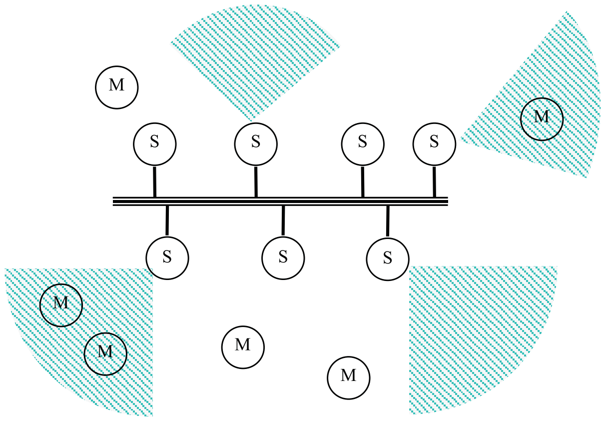

Fig. 2. System with 7 *SHs*, 4 gateways and 6 *MHs*

and to a wireless link, that covers a spatially limited *cell* nearby the gateway. MHs may move and communicate through wireless links. A gateway may broadcast messages to all MHs in its cell and send messages to a specific MH in its cell. A MH may only exchange messages with the gateway of the cell where it happens to be located. To take into account physical obstructions, communication within a cell is FIFO-ordered but messages may be lost. There is no support for routing messages to a specific MH, i.e., we do not rely on Mobile IP for the reasons detailed in [2]. MHs may roam in areas that are not covered by any cell. Finally, it is assumed that hosts never crash.

The protocol supports multicast communication within a static group of processes (called group members). For ease of presentation, we shall refer to a single group in which all group members run on MHs and such that distinct members run on distinct MHs. We shall associate a different meaning with the terms "receiving" and "delivering" a message. A computer C *receives* a message msg when msg arrives at the protocol layer at C. Upon receiving msg, the protocol may perform any of: (i) buffer msg; (ii) discard msg; (iii) pass msg up to the application. In case (iii), C is said to *deliver* msg.

The protocol works as follows. A dedicated SH acts as the *coordinator*, denoted as `S_C`, and its identity is known to all group members. A group member generates a multicast by sending a `new` message containing the payload to `S_C`, that processes incoming `new` messages in sequence. `S_C` constructs a message tagged `normal` containing the payload and an increasing group-wide sequence number. `S_C` then transmits the resulting message to gateways through a FIFO-multicast. Gateways broadcast this message to group members in the respective cells.

Due to its movement across cells and uncovered areas, any group member m could receive duplicates or could miss multicasts. By maintaining a history of the received sequence numbers, m discards duplicates in the former case and sends to the gateway a proper `nack` message in the latter (e.g., upon receiving an out-of-order message). Upon receiving a `nack`, the gateway will relay to m a copy of the missing multicasts, through a sequence of `transfer` messages. Each gateway stores a copy of each multicast previously sent until it knows that the multicast has been delivered by every group member, which ensures that each gateway may generate autonomously the required sequence of `transfer` messages. Moreover, the gateway periodically (actually when a timeout occurs) broadcasts the last `normal` message received from the coordinator [2].

Note that the protocol does not use any notion of hand-off (unlike the protocol in [1]), i.e., it does not require any data exchange between the old gateway and the new one when a MH moves from one cell to another. Each gateway manages cell switchings autonomously, without interacting with other SHs.

We shall omit the discussion of how gateways become aware that a certain message has been delivered by every group member, which is necessary to garbage-collect unnecessary messages. Essentially, a `nack`

[2] Without this mechanism if the last message gets lost it could not be detected (in case no further `normal` messages are ever generated).

carries an indication of which messages have been delivered by the group member that originates the `nack`. This information is extracted by the gateway and propagated as appropriate.

4.1 Properties of the protocol

The protocol was designed to guarantee the following properties:

Total Order (P_1) If any two group members m_1 and m_2 deliver multicasts msg_1 and msg_2, then they deliver these multicasts in the same order, i.e. either they both deliver msg_1 before msg_2, or they both deliver msg_2 before msg_1.

No Duplicate (P_2) No group member delivers duplicate multicasts, i.e. duplicates are discarded.

Non Triviality (P_3) Each group member m delivers each multicast under reasonable assumptions, as follows: m stops delivering messages if: (i) m starts entering and leaving cells so quickly that its messages never arrive to any gateway or messages from gateways are systematically lost; and (ii) this pattern of movements persists forever. We assume that the wireless link does not loose all messages mentioned in (i).

Integrity (P_4) Any multicast delivered by a group member has been originated by a group member.

Properties P_1 and P_2 are the essence of the protocol: if group members deliver multicasts then they deliver them in the required total order and with no duplicates. However, these properties are clearly not sufficient since they would be satisfied even by a protocol in which group members do not deliver any multicast. Similar trivial solution is ruled out by the property P_3, i.e. group members indeed deliver multicasts. Finally, P_4 guarantees that multicasts are not generated capriciously.

5 Protocol specification

Coordinator

The coordinator `S_C` maintains a variable `seq` containing the sequence number of the last `normal` message sent. Initially `seq` is 0. `S_C` may receive messages of the form `new(b)`. One such message `msg` informs `S_C` that a sending group member wants to multicast `b`. Actions performed by the coordinator are described by the CCS process `S_C(seq)`: upon receiving `msg`, `S_C`: (1) builds a message `msg1 = normalCG(g,b,seq+1)` (`g` represents a gateway identifier); (2) transmits `msg1` to all gateways `gi` through a FIFO-multicast; (3) increments `seq`. (Note: CWB-NC represents output actions as `'a` instead of $\overline{\mathtt{a}}$. Moreover the process `S_C(seq)` can be regarded as shorthand for the indexed constant process `S_C_seq` and the action `normalCG(gi,b,seq+1)` as shorthand for the indexed action `normalCG_gi_b_seq+1`. Similarly for the action `new(b)`).

```
S_C(seq) = new(b).'normalCG(g1,b,seq+1)...'normalCG(gn,b,seq+1).S_C(seq+1)
```

Gateway

Each gateway `g` maintains the following data structures: **(i)** `l`: a list of `normal` messages recently received from `S_C`; **(ii)** `seq`: a variable containing the sequence number of the last `normal` message received. Initially, `l` is empty and `seq` is 0. Actions performed by each gateway are described by the CCS process `G(g,l,seq)` and discussed below. `g` may receive `normal` messages from the coordinator. `g` simply broadcasts the message in the cell, appends it to `l` and increments `seq`. `g` may receive messages of the form `nack(m,g,s1,s2)`. One such message informs `g` that the sending group member `m` has delivered all multicasts with sequence number up to `s1` (excluded) but that it missed those with sequence number `s` $\in$ [`s1`, `s2`]. Accordingly, `g` starts transferring to `m` a copy of the missing multicasts. Moreover, `g` repeatedly (actually when a timeout occurs) broadcasts the last message that has received.

```
G(g,l,seq) = normalCG(g,b,seq+1).BROADCAST(normal,g,l,b,seq+1) +
             nack(m,g,s1,s2).TRANSFER(m,g,s1,s2)+
             timeout.BROADCAST(normal,g,l,b,seq)
```

The broadcast of normal messages, when only two group members are considered, is modeled as:

```
BROADCAST(normal,g,l,b,seq+1) =

'normal(m1,g,b,seq+1).('normal(m2,g,b,seq+1).G(g,l::b,seq+1) + G(g,l::b,seq+1)) +
'normal(m2,g,b,seq+1).('normal(m1,g,b,seq+1).G(g,l::b,seq+1) + G(g,l::b,seq+1)) +
G(g,l::b,seq+1)
```

The transfer of missing multicasts (process `TRANSFER(m,g,s1,s2)`) can be defined in a similar way.

Group member

Each group member `m` maintains the data structure `mystable`, i.e. a variable containing the sequence number of the last multicast delivered. Initially, `mystable` is 0. Actions performed by each group member are described by the CCS process `M(m,g,mystable)` and discussed below. A group member may receive messages from the application (`request(m,b)`) to multicast into the group. Such messages are sent by the group member as `new` messages to the coordinator and are then broadcast by the gateways to all the group members in the cell as `normal` messages. A group member receives multicasts by means of messages tagged either `normal` or `transfer`. A multicast is delivered only if it carries a sequence number `s` such that `s=mystable+1`. Upon delivering a multicast, the group member increments `mystable`. If the sequence number of the received message is `s<mystable+1` the message is discarded since it is a duplicate message. If condition `s>mystable+1` occurs the group member realizes that it has missed all messages in the range `[mystable+1,s-1]`. In this case, the message is discarded and the group member notifies the gateway `g` by means of a message `msg = nack(m,g,mystable+1,s)`. The fact that the group member has sent a `nack` does not imply that it will receive all required multicasts, because some `transfer` messages could be lost. These messages, if any, will be requested again by further `nack` messages. Finally, a group member may move: this is expressed by action `move`. In the following specification we use the shorthand `if b then p` to express that the process behaves as the process `p` if the boolean expression `b` evaluates to true.

```
M(m,g,mystable) =

     request(m,b).'new(b).M(m,g,mystable) +
     normal(m,g,b,s).
         {if (s=mystable+1) then 'deliver(m,b).M(m,g,mystable+1);
          if (s>mystable+1) then 'nack(m,g,mystable+1,s).M(m,g,mystable);
          if (s<mystable+1) then  M(m,g,mystable)} +
     transfer(m,g,b,s).
         {if (s=mystable+1) then 'deliver(m,b).M(m,g,mystable+1);
          if (s>mystable+1) then 'nack(m,g,mystable+1,s).M(m,g,mystable);
          if (s<mystable+1) then  M(m,g,mystable)} +
     move.('moved(m,g1,mystable).M(m,g1,mystable) + .. +
           'moved(m,gn,mystable).M(m,gn,mystable) {with gi<>g, for all i})
```

The CCS process `prot` describes at the highest level the protocol, with `r` gateways and `k` group members. We suppose that all group members are initially located in the cell of gateway `1`.

`prot` = (`S_C(0)` | `G(1,lambda,0)` | ... | `G(r,lambda,0)` | `M(1,1,0)` | ... |`M(k,1,0)` | `env`)`\internals`

where:

– the set `internals` contains all actions used for the communication among the processes of the parallel composition
– `env` represents the environment, which produces the messages to be delivered and the timeout for the gateways.

6 Protocol Verification

We now apply our methodology to verify the protocol. First we express the required properties of the protocol using the selective mu-calculus. Then we apply our tool to transform the specification into a smaller one, where the reduction is driven by the actions occurring in the formulae. Finally, we check the properties on the reduced specification.

The properties described in Section 4.1 can be expressed with the selective mu-calculus in the following way, where $\mathtt{msg}, \mathtt{msg}'$ are messages and $\mathtt{m}, \mathtt{m}'$ are group members:

Total Order

$$P_1 = [\mathtt{'deliver(m,msg)}]_{\{\mathtt{'deliver(m',msg)}\}}[\mathtt{'deliver(m,msg')}]_{\{\mathtt{'deliver(m',msg)}\}}\ P$$
$$\wedge$$
$$[\mathtt{'deliver(m,msg)}]_{\{\mathtt{'deliver(m',msg)}\}}\ \langle\mathtt{'deliver(m',msg)}\rangle_{\emptyset}\mathtt{tt}$$

where $P = [\mathtt{'deliver(m',msg')}]_{\{\mathtt{'deliver(m',msg)}\}}\mathtt{ff}$.

$$\mathcal{O}(P_1) = \rho_1 = \bigcup_{\mathtt{i}\in\{\mathtt{m},\mathtt{m}'\},\mathtt{j}\in\{\mathtt{msg},\mathtt{msg}'\}} \{\mathtt{'deliver(i,j)}\}$$

The meaning of P_1 is: "if `m` delivers first `msg` and then $\mathtt{msg}'$, then $\mathtt{m}'$ cannot deliver $\mathtt{msg}'$ if it has not delivered `msg` and if `m` delivers `msg` then also $\mathtt{m}'$ can deliver `msg`".

No Duplicate

$$P_2 = [\mathtt{'deliver(m,msg)}]_{\emptyset}[\mathtt{'deliver(m,msg)}]_{\emptyset}\mathtt{ff}$$

$$\mathcal{O}(P_2) = \rho_2 = \{\mathtt{'deliver(m,msg)}\}$$

The meaning of P_2 is: "when an action `'deliver(m,msg)` is performed, the same action cannot be performed any more".

Non Triviality

$$P_3 = \nu\mathtt{Z}.(<'\ \mathtt{deliver(m,msg)} >_{\emptyset} \mathtt{tt} \wedge [-\mathtt{'deliver(m,msg)}]_{\mathcal{A}}\mathtt{Z})$$

$$\mathcal{O}(P_3) = \rho_3 = \mathcal{A}$$

The meaning of P_3 is: "in each state, reached by performing an action different from `'deliver(m,msg)`, it is possible to perform `'deliver(m,msg)`".

Integrity

$$P_4 = [\mathtt{'deliver(m,msg)}]_{\{\mathtt{send(msg)}\}}\mathtt{ff}$$

$$\mathcal{O}(P_4) = \rho_4 = \{\mathtt{'deliver(m,msg)}, \mathtt{send(msg)}\}$$

The meaning of P_4 is: "it is not possible to perform an action `'deliver(m,msg)` if an action `send(msg)` has not been previously performed". Note that the action `send(msg)` is the message to be delivered produced by the environment (process `env` of Section 5).

Note that the property P_3 does not mean that the message is eventually delivered, but only that the possibility of delivering is never lost. In fact, it is possible that a message is not delivered if either the group member enters and leaves cells too quickly, or messages from gateways are systematically lost. Property P_3 ensures that, if the above situation does not persist forever, the message is eventually delivered.

Moreover, the set of actions ρ_3 occurring in the formula P_3 coincides with the whole set $\mathcal{A}$. This means that the transformed program is equal to the original one and in this case we obtain no advantage. Some reductions could be obtained if we instead checked the following property, which is weaker than P_3:

Consistency

$$P_5 = [\texttt{'deliver(m,msg)}]_{\{\texttt{'deliver(m',msg)}\}}\langle \texttt{'deliver(m',msg)}\rangle_{\emptyset}\texttt{tt}$$

$$\mathcal{O}(P_5) = \rho_5 = \{\texttt{'deliver(m,msg)},\texttt{'deliver(m',msg)}\}$$

The meaning of P_5 is: "if `m` delivers `msg` then `m'` can always deliver `msg`" (`m'` actually delivers `msg` under the reasonable assumptions specified for P_3 in Section 4.1, i.e. group members do not always enter and leave cells too quickly, and messages from gateways are not systematically lost). Note that the set of actions ρ_5 occurring in the formula P_5 is a small subset of the whole set $\mathcal{A}$.

The above properties can be expressed for any number of gateways, group members, and messages. We have instantiated the specification with two gateways, `g1` and `g2` and two group members, `m1` and `m2`. To make the experiments we used the CWB-NC tool on a PC based on a 350 MHz Pentium II processor with 256 Mbytes of memory and FREEBSD operating system.

	$S(\texttt{prot})$		$S(\texttt{Tprot}_{\rho_1})$			$S(\texttt{Tprot}_{\rho_2})$		
n	states	transitions	states	transitions	state space reduction %	states	transitions	state space reduction %
2	13803	63055	6459	29795	52.8%	3880	18279	71.1%
3	-	-	105189	527875	-	63596	331124	-

Table 4. *Results for the proposed protocol (I)*

	$S(\texttt{prot})$		$S(\texttt{Tprot}_{\rho_4})$			$S(\texttt{Tprot}_{\rho_5})$		
n	states	transitions	states	transitions	state space reduction %	states	transitions	state space reduction % %
2	13803	63055	3884	18291	71.1%	4730	22082	65.1%
3	-	-	63600	331136	-	72390	373578	-

Table 5. *Results for the proposed protocol (II)*

Tables 4 and 5 show the number of states and transitions of $S(\texttt{prot})$ and of $S(\texttt{Tprot}_{\rho_i})$, $i \in \{1,2,4,5\}$, for different values of `n`, where `n` is the number of messages and $\texttt{Tprot}_{\rho_i}$ is the reduced CCS process obtained by applying the transformation rules to `prot` with ρ_i, for each $i \in \{1,2,4,5\}$. Note that for `n=3` the CWB-NC tool was not able to build the complete transition system, while the reduced system could be built and so the properties were proved to hold. If `n=4` also our tool is not able to produce the reduced transition systems. However, this is not an intrinsic limitation of our methodology. By using a more powerful computer it could be possible to consider a greater number of messages. The state space reduction implies a corresponding reduction of the verification time: Table 6 shows the time employed by the CWB-NC to check the formulae and the time reduction obtained with our methodology.

	standard transition system	reduced transition system	time reduction %
P_1	63s	27.7s	56%
P_2	62.4s	15.4s	75.3%
P_3	62.6s	62.6s	0%
P_4	62.2s	15.6s	74.9%
P_5	61.1s	19.9s	67.4%

Table 6. *Verification time of P_1, P_2, P_3, P_4 and P_5, for $n = 2$*

Finally, Table 7 shows the time spent by the CWB-NC to generate $S(\texttt{prot})$ and $S(\texttt{Tprot}_{\rho_i})$, for each $i \in \{1, 2, 4, 5\}$. Note that in some cases we can reach a reduction of 75 per cent.

In general, the usefulness of our tool depends both on the number of actions occurring in the formula and on the structure of the formula to be checked. Following the classification in [28] of the properties as *liveness, weak liveness, safety and weak safety* properties, in [6] it is shown that the properties we manage successfully can be informally characterized as follows:

- $\langle K, R \rangle$*-weak liveness properties*: properties requiring that there exists a finite path, composed only by actions in K, and ending with an action in R, which leads to a satisfactory state.
- $\langle K, R \rangle$*-safety properties*: properties requiring that every finite path, composed only by actions in K, and ending with an action in R, leads to a satisfactory state.

For instance, integrity property (i.e. P_4) is a $\langle K, R \rangle$-safety property, where $R = \{'\texttt{deliver}(\texttt{m}, \texttt{msg})\}$ and $K = \mathcal{A} - \{'\texttt{deliver}(\texttt{m}, \texttt{msg}), \texttt{send}(\texttt{msg})\}$.

On the other hand, we must consider the fact that the reduction we obtain is not made once-and-for-all, but may be different for different formulae. Thus we must also consider in the verification time the time employed to build the different reduced transition systems (see Table 7). This drawback can be in some cases avoided by taking a unique set of actions for different formulae, which is the union of the respective sets of occurring actions. In fact in [6] it is proved that the truth value of a formula is the same if we consider a super-set of the occurring actions. In the protocol case, we can check both P_2 and P_4 on $S(\texttt{Tprot}_{\rho_4})$, since $\rho_2 \subseteq \rho_4$.

n	$S(\texttt{prot})$	$S(\texttt{Tprot}_{\rho_1})$		$S(\texttt{Tprot}_{\rho_2})$		$S(\texttt{Tprot}_{\rho_4})$		$S(\texttt{Tprot}_{\rho_5})$	
	time	time	time reduction %	time	time reduction %	time	time reduction %	time	time reduction %
2	58.3s	26.1s	55.2%	14.8s	74.6%	15s	74.2%	19.1s	67.2%
3	-	1013.5s	-	623.5s	-	633.1s	-	737.6s	-

Table 7. *Time spent to generate the standard and reduced transition systems*

7 Related work and conclusions

In this paper we have presented the formal verification of a multicast protocol for mobile computing. The protocol was chosen as a case study to demonstrate the usefulness of a methodology which reduces transition systems for model checking temporal logic formulae. However, the methodology is quite general and thus can be applied to other protocols. For example, it is possible to model other multicast protocols for mobile computing (e.g., the protocol in [1] which provides FIFO ordering of messages and do not use the coordinator) as well as protocols which do not take mobility into consideration. The method is based on the selective mu-calculus logic, which has the property that each formula allows us to immediately point out the parts of the transition system that do not alter the truth value of the formula itself.

We remark that the degree of reduction we obtain with our approach essentially depends on the number of actions syntactically occurring in the modal operators of the formula to be checked. When these actions are almost the whole set $\mathcal{A}$ of actions, we do not obtain significant reductions.

A main difference of our approach from other existing ones [12, 13] is that the reduction is not provided by the user (often based on an informal reasoning and proved correct case by case), but it is completely automatic and transparent to the user. In fact, the reductions are driven by the formulae to be checked.

The works [4, 19] present methods for taking a given formula into account in constructing reduced transition systems: the reduction preserves the truth values of the formula only if the properties are expressed by formulae obeying some restrictions (for example avoiding some operators of the logic CTL), or requiring transition systems of a particular kind. Moreover, in both cases, the algorithms used to obtain the reduced transition systems are not trivial.

The works [21, 26, 29] follow the partial order approach to model checking, in which only a representative is considered among all interleavings of actions generated by a parallel composition. The properties well handled by these approaches do not concern precedence relations between actions, and can be profitably used to prove, for example, deadlock freedom. In our approach, properties concerning precedence relations between actions are in general described by formulae which induce a consistent reduction of the state space of the transition system. On the contrary, the formula describing deadlock freedom induces no reduction, since it involves all actions. Thus the two approaches can be considered as complementary.

In [15, 22] modular approaches have been proposed to attack the state explosion problem. These approaches are based on dividing the verification task into simpler tasks, exploiting the natural decomposition of complex systems into processes. Following a modular approach, in [23] the specification and verification of fault-tolerant broadcasts has been presented.

Finally, all the verification systems which base their behavior on the analysis of transition systems can profit from our formula-based reductions. In particular, our approach can be integrated with the on-the-fly methodology [20], or with the tableau-based approach [17], or with the automata-theoretic approach [10]. For example, given a formula of the selective mu-calculus with occurring actions ρ, we can on-the-fly verify it during the generation of the reduced transition system, instead of during the generation of the standard one.

References

1. A. Acharya, B. Badrinath. A framework for delivering multicast messages in networks with mobile hosts. *ACM/Baltzer Mobile Networks and Applications, 1(2)*, 1996. 199-219.
2. G. Anastasi, A. Bartoli, F. Spadoni. Group Multicast in Distributed Mobile Systems with Unreliable Wireless Network. In *Proceedings of the 18th IEEE Symposium on Reliable Distributed Systems (SRDS'99)*, Lausanne (CH), October 18-21, 1999.
3. G. Anastasi, A. Bartoli, F. Spadoni. A Reliable Multicast Protocol for Distributed Mobile Systems: Design and Evaluation (extended version). *MOSAICO Project Technical Report PI-DII/2/99*, September 1999. Available at: http://www.iet.unipi.it/~anastasi/papers/tr99- 2.pdf.gz.
4. A. Aziz, T.R. Shiple, V. Singhal, A.L. Sangiovanni-Vincentelli. Formula-Dependent Equivalence for Compositional CTL Model Checking. In *Proceedings of Workshop on Computer Aided Verification (CAV'94), Lecture Notes in Computer Science* 818, 1994. 324–337.

5. R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. Selective Mu-Calculus: New Modal Operators for Proving Properties on Reduced Transition Systems. In *Proceedings of FORTE X/PSTV XVII '97*. Chapman & Hall, 1997. 519-534.
6. R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. Selective Mu-Calculus and Formula-based Equivalence of Transition Systems. *Journal of Computer and System Sciences*, 59(3), (1999).
7. R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. Constructing Reduced Models for Temporal Properties Verification. *MOSAICO Project Technical Report PI-DII/5/98*, July 1998.
8. R. Barbuti, N. De Francesco, A. Santone, G. Vaglini. LORETO: a Tool for Reducing State Explosion in Verification of Lotos Programs. *Software-Practice and Experience*, 29(12), (1999). 1123-1147.
9. A. Bartoli. Group-Based Multicast and Dynamic Membership in Wireless Networks with Incomplete Spatial Coverage. *ACM/Baltzer Mobile Networks and Applications*, 3(2), (1998). 175–188.
10. O. Bernholtz, M.Y. Vardi, P. Wolper. An Automata-Theoretic Approach to Branching-Time Model Checking. In *Proceedings of Workshop on Computer Aided Verification (CAV'94), Lecture Notes in Computer Science* 818, 1994. 142–155.
11. T. Bolognesi, E. Brinksma. Introduction to ISO Specification Language LOTOS. *Comp. Networks and ISDN Systems*, 14, (1987). 25-59.
12. G. Bruns. A Case Study in Safety-Critical Design. In *Proceedings of Workshop on Computer Aided Verification (CAV'92), Lecture Notes in Computer Science* 663, 1992. 220–233.
13. G. Bruns. A Practical Technique for Process Abstraction. In *Proceedings of International Conference on Concurrency Theory (CONCUR'93), Lecture Notes in Computer Science* 714, 1993. 37–49.
14. E.M. Clarke, E.A. Emerson, A.P. Sistla. Automatic Verification of Finite-state Concurrent Systems using Temporal Logic Verification. *ACM Transactions on Programming Languages and Systems*, 8, (1986). 244–263.
15. E.M. Clarke, D.E. Long, K.L. McMillan. Compositional Model Checking. In *Proceedings of the Fourth Annual IEEE Symposium on Logic in Computer Science*, 1989. 353-362.
16. R. Cleaveland, S. Sims. *The NCSU Concurrency Workbench.* In Proceedings of Workshop on Computer Aided Verification (CAV'96), Lecture Notes in Computer Science 1102, 1996. 394–397.
17. R. Cleveland. Tableau-based Model Checking in the Propositional Mu-Calculus. *Acta Informatica, 27*, (1990). 725–747.
18. The Concurrency Workbench of North Carolina home page.
URL http://www4.ncsu.edu/eos/users/r/rance/WWW/ncsu-cw.html.
19. D. Dams, O. Grumberg, R. Gerth. Generation of Reduced Models for Checking Fragments of CTL. In *Proceedings of Workshop on Computer Aided Verification (CAV'93), Lecture Notes in Computer Science* 697, 1993. 479–490.
20. J.C. Fernandez, L. Mounier. "On th Fly" Verification of Behavioural Equivalences and Preorders. In *Proceedings of the Third International Conference on Computer-Aided Verification, Lecture Notes in Computer Science* 575, 1991. 181–191.
21. P. Godefroid. Partial-Order Methods for the Verification of Concurrent Systems. *Lecture Notes in Computer Science* 1032, 1996.
22. O. Grumberg, D.E. Long. Model Checking and Modular Verification. *ACM Transactions on Programming Languages and Systems, 16(3)*, 1994. 843-871.
23. V. Hadzilacos, S. Toueg. A Modular Approach to Fault-Tolerant Broadcasts and Related Problems. Technical Report TR94-1425. Department of Computer Science, Cornell University, Ithaca NY. May 1994.
24. D. Kozen. Results on the Propositional Mu-Calculus. *Theoretical Computer Science* 27, 1983. 333–354.
25. R. Milner. *Communication and Concurrency.* Prentice-Hall, 1989.
26. D. Peled. All from One, One for All, on Model-Checking Using Representatives. In *Proceedings of the Fifth International Conference on Computer-Aided Verification(CAV'93), Lecture Notes in Computer Science* 679, 1993. 409–423.
27. SICStus Prolog User's Manual. Swedish Institute of Computer Science. Release 3.7.1, October 1998. Swedish Institute of Computer Science. http://www.sics.se/isl/sicstus.html.
28. C. Stirling. An Introduction to Modal and Temporal Logics for CCS. In *Concurrency: Theory, Language, and Architecture, Lecture Notes in Computer Science* 391, 1989.
29. A. Valmari. A Stubborn Attack on State Explosion. In *Proceedings of International Conference on Computer-Aided Verification (CAV'90), Lecture Notes in Computer Science* 531, 1990. 156–165.