TLP **6** (3): 321–353, 2006. © 2006 Cambridge University Press
doi:10.1017/S1471068406002663 Printed in the United Kingdom

Parametric Verification of a Group Membership Algorithm

AHMED BOUAJJANI
LIAFA, *University of Paris 7, Case 7014, 2 place Jussieu, 75251 Paris 5, France*
(*e-mail:* abou@liafa.jussieu.fr)

AGATHE MERCERON
LIAFA *and* ESILV-GI, *Technical University Leonard de Vinci, 92916 Paris La Défense, France*
(*e-mail:* Agathe.Merceron@liafa.jussieu.fr)

submitted 16 January 2004; revised 17 March 2005; accepted 10 May 2005

Abstract

We address the problem of verifying clique avoidance in the TTP protocol. TTP allows several stations embedded in a car to communicate. It has many mechanisms to ensure robustness to faults. In particular, it has an algorithm that allows a station to recognize itself as faulty and leave the communication. This algorithm must satisfy the crucial 'non-clique' property: it is impossible to have two or more disjoint groups of stations communicating exclusively with stations in their own group. In this paper, we propose an automatic verification method for an arbitrary number of stations N and a given number of faults k. We give an abstraction that allows to model the algorithm by means of unbounded (parametric) counter automata. We have checked the non-clique property on this model in the case of one fault, using the ALV tool as well as the LASH tool.

KEYWORDS: Formal verification, fault-tolerant protocols, parametric counter automata, abstraction

1 Introduction

The verification of complex systems, especially of software systems, requires the adoption of powerful methodologies based on combining, and sometimes iterating, several analysis techniques. A widely adopted approach consists in combining abstraction techniques with verification algorithms (e.g., model-checking, symbolic reachability analysis (Graf and Saidi 1997; Abdulla *et al.* 1999, Saidi and Shankar 1999)). In this approach, non-trivial abstraction steps are necessary to construct faithful abstract models (typically finite-state models) on which the required properties can be automatically verified. The abstraction steps can be extremely hard to carry out depending on how restricted the targeted class of abstract models is. Indeed, many aspects in the behavior of complex software systems cannot (or can hardly) be captured using finite-state models. Among these aspects, we can mention:

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

(1) the manipulation of variables and data-structures (counters, queues, arrays, etc.) ranging over infinite domains, and (2) parameterization (e.g., sizes of the data structures, the number of components in the system, the rates of errors/faults/losses, etc.). For this reason, it is often needed to consider abstraction steps which yield infinite-state models corresponding to extended automata, i.e., a finite-control automata supplied with unbounded data-structures (e.g., timed automata, pushdown automata, counter automata, FIFO-channel automata, finite-state transducers, etc.) (Abdulla *et al.* 1999). Then, symbolic reachability analysis algorithms (Cousot and Halbwachs 1978; Boigelot and Wolper 1994; Bouajjani *et al.* 1997; Bouajjani and Habermehl 1997; Kesten *et al.* 1997; Bultan *et al.*; Wolper and Boigelot; Bouajjani *et al.* 2000; Annichini *et al.* 2000; Abdulla and Jonsson 2001a; Abdulla and Jonsson 2001b) can be applied on these (abstract) extended automata-based models in order to verify the desired properties of the original (concrete) system. Of course, abstraction steps remain non-trivial in general for complex systems, even if infinite-state extended automata are used as abstract models.

In this paper, we consider verification problems concerning a protocol used in the automotive and aerospace industry. The protocol, called TTP/C, was designed at the Technical University of Vienna to allow communication between several devices (micro-processors) embedded in a car or plane, whose function is to control the safe execution of different driving actions (Kopetz and Grünsteidl 1999; Kopetz 1999).

The protocol involves many mechanisms to ensure robustness to faults. In particular, the protocol involves implicit and explicit mechanisms which allow to discard devices (called stations) which are (supposed to be) faulty. This mechanism must ensure the crucial property: *all active stations form one single group of communicating stations, i.e., it is impossible to have two (or more) disjoint groups of active stations communicating exclusively with stations in their own group.*

Actually, the algorithm is very subtle and its verification is a real challenge for formal and automatic verification methods. Roughly, it is a parameterized algorithm for N stations arranged in a ring topology. Each of the stations broadcasts a message to all stations when it is its turn to emit. The turn of each station is determined by a fixed time schedule. Stations maintain informations corresponding to their view of the global state of the system: a membership vector, consisting of an array with a parametric size N, telling which stations are active. Stations exchange their views of the system and this allows them to recognize faulty stations. Each time a station sends a message, it sends also the result of a calculation which encodes its membership vector. Stations compare their membership vectors to those received from sending stations. If a receiver disagrees with the membership vector of the sender, it counts the received message as incorrect. If a station disagrees with a majority of stations (in the round since the last time the station has emitted), it considers itself as faulty and leaves the active mode (it refrains from emitting and skips its turn). Stations which are inactive can return later to the active mode (details are given in the paper). Besides the membership vector, each station s maintains two integer counters in order to count in the last round (since the previous emission of the station s) (1) the number of stations which have emitted and from which s has received a correct message with membership vector equal to its own vector at that

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

moment (the stations may disagree later concerning some other emitting station), and (2) the number of stations from which s received an incorrect message (the incorrect message may be due to a transmission fault or to a different membership vector). The information maintained by each station s depends tightly on its position in the ring relatively to the positions of the faulty stations and relatively to the stations which agree/disagree with s w.r.t. each fault.

The proof of correctness of the algorithm and its automatic verification are far from being straightforward, especially in the parametric case, i.e., for any number of stations, and any number of faults.

The first contribution of this paper is to prove that the algorithm stabilizes to a state where all membership vectors are equal after precisely two rounds from the occurrence of the last fault in any sequence of faults. The proof is given for the general case where re-integrating stations are allowed. To guarantee stabilization after k-faults in the case of re-integration, we propose an algorithm slightly different from the one presented by Kopetz and Grünsteidl (1999), which guarantees stabilization in the case of 1 fault only. The generalization to k faults makes an assumption on the failure model (made explicit in section 2) that may not be realistic for a particular kind of messages called N-frames (Kopetz and Grünsteidl 1999).

Then, we address the problem of automatically verifying the algorithm. We prove that, for every fixed number of faults k, it is possible to construct an abstraction of the algorithm (parameterized by the number of stations N) by means of a parametric counter automaton. This result is surprising since (1) it is not easy to abstract the information related to the topology of the system (ordering between the stations in the ring), and (2) each station (in the concrete algorithm) has local variables ranging over infinite domains (two counters and an array with parametric bounds). The difficulty is to prove that it is possible to encode the information needed by all stations by means of a finite number of counters. Basically, this is done as follows: (1) We observe that a sequence of faults induces a partition of the set of active stations (classes correspond to stations having the same membership vector) which is built by successive refinements: Initially, all stations are in the same set, and the occurrence of each fault has the effect of splitting the class containing the faulty station into two subclasses (stations which recognizes the fault, and the other ones). (2) We show that there is a partition of the ring into a finite number of regions (depending on the positions of the faulty stations) such that, to determine at any time whether a station of any class can emit, it is enough to know how many stations in the different classes/zones have emitted in the last two rounds. This counting is delicate due to the splitting of the classes after each fault.

Finally, we show that, given a counter automaton modeling the algorithm, the stabilization property (after two rounds following the last fault) can be expressed as a constrained reachability property (in CTL with Presburger predicates) which can be checked using symbolic reachability analysis tools for counter automata (e.g., ALV (Bultan and Yavuz-Kahveci 2001) or LASH (LASH 2002)). We have experimented this approach in the case of one fault. We have built a model for the algorithm in the language of ALV, and we have been able to verify automatically

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

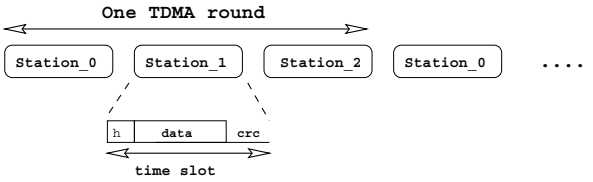


Fig. 1. A TDMA round for three stations.

that it converges to a single clique after precisely two rounds from the occurrence of the fault. Actually, we have provided a refinement of the abstraction given in the general case which allows to build a simpler automaton. This refinement is based on properties specific to the 1 fault case that has been checked automatically using ALV.

The paper is organized as follows. Section 2 presents the protocol. In Section 3, we prove the crucial non-clique property for n stations: the stations that are still active do have the same membership vector at the end of the second round following fault k. Considering the 1 fault case, section 4 presents how to abstract the protocol parameterized by the number of stations n as an automaton with counters that can be symbolically model checked. Section 5 generalizes the approach for a given number of faults k. Section 6 concludes the paper. A preliminary version of this paper has appeared in (Bouajjani and Merceron 2002).

2 Informal description of the protocol

TTP is a time-triggered protocol. It has a finite set S of N stations and allows them to communicate via a shared bus. Messages are broadcast to all stations via the bus. Each station that participates in the communication sends a message when it is the *right* time to do so. Therefore, access to the bus is determined by a time division multiple access (TDMA) schema controlled by the global time generated by the protocol. A TDMA round is divided into *time slots*. The stations are statically ordered in a ring and time slots are allocated to the stations according to their order. During its time slot, a station has exclusive message sending rights. A TDMA round for three stations is shown in Figure 1. When one round is completed, a next one takes place following the same pattern.

TTP is a fault-tolerant protocol. Stations may fail while other stations continue communicating with each other. TTP provides different services to ensure robustness to faults, such as replication of stations, replication of communication channels, bus guardian, fault-tolerant clock synchronization algorithm, implicit acknowledgment, clique avoidance mechanism, (Kopetz and Grünsteidl 1999; Kopetz 1999; Bauer and Paulitsch 2000). Several classes of faults are distinguished. Steiner *et al.* (2004), for example, focuses on faults that may appear at startup. In this paper, we focus on asymmetric faults. A symmetric fault occurs when a station is *send faulty*, i.e., no other station can receive it properly, or *receive faulty*, i.e., it cannot receive properly any message. Asymmetric faults occur when an emitting station is received properly

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

by more than 1 station, but less then all stations. We allow asymmetric faults to occur and consider symmetric faults as a special case of asymmetric faults. For the protocol to work well, it is essential that (asymmetric) faults do not give rise to cliques. In Kopetz and Grünsteidl (1999) and Kopetz (1999), *cliques* are understood as disjoint sets of stations communicating exclusively with each other. In this paper, we focus on implicit acknowledgment and clique avoidance mechanism, to be introduced shortly, and show that they prevent the formation of different cliques, clique is cast in its graph theoretical meaning.

2.1 Local information

When it is working or in the `active` state, a station sends messages in its time slot, listens to messages broadcast by other stations and carries local calculations. Each station s stores locally some information, in particular a *membership vector* m_s and two counters, $CAcc_s$ and $CFail_s$. A *membership vector* is an array of booleans indexed by S, the set formed by the N stations. It indicates the stations that s receives correctly (in a sense that will be made precise below). If s received correctly the last message, also called *frame*, sent by s', then $m_s[s'] = 1$, otherwise $m_s[s'] = 0$. A sending station is supposed to receive itself properly, thus $m_s[s] = 1$ for a working station s. The counters $CAcc_s$ and $CFail_s$ are used as follows. When it is ready to send, s resets $CAcc_s$ and $CFail_s$ to 0. During the subsequent round, s increases $CAcc_s$ by 1 each time it receives a correct frame (this includes the frame it is sending itself) and it increases $CFail_s$ by 1 each time it receives an incorrect frame. When no frame is sent (because the station that should send is not working), neither $CFail_s$ nor $CAcc_s$ are increased.

2.2 Implicit acknowledgment

Frames are broadcast over the bus to all stations but they are not explicitly acknowledged. TTP has *implicit acknowledgment*. A frame is composed of a header, denoted by `h` in Figure 1, a data field, denoted by `data` and a CRC field denoted by `crc`. The data field contains the data, like sensor-recorded data, that a station wants to broadcast. The CRC field contains the calculation of the Cyclic Redundancy Check done by the sending station. CRC is calculated over the header, the data field and the individual membership vector. When station s is sending, it puts in the CRC field the calculation it has done with its own membership vector m_s. Station s' receiving a frame from station s recognizes the frame as *valid* if all the fields have the expected lengths. If the frame is valid, station s' performs a CRC calculation over the header and the data field it has just received, and its own membership vector $m_{s'}$. It recognizes the frame as *correct* if it has recognized it as valid and its CRC calculation agrees with the one put by s in the CRC field. Therefore, a correct CRC implies that sender s and receiver s' have the same membership vector.

We also assume a converse: if s and s' do not have the same membership vector, the CRC is not correct. This assumption may be strong and may not be met by

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

special messages of TTP called N-frames. However, it is valid for X-frames and I-frames.

During normal operation, the sender s performs two CRC checks over the frame received from its first successor s':

`CheckIa` CRC calculation with $m_s[s] = 1$ and $m_s[s'] = 1$.

`CheckIb` CRC calculation with $m_s[s] = 0$ and $m_s[s'] = 1$.

If the CRC `CheckIa` is correct, s knows that s and s' have identical membership vectors, so it *implicitly* deduces that its successor s' has received its frame. Thus s assumes that it is not faulty, $m_s[s]$ remains 1 and `CheckIb` is discarded. It increases $CAcc_s$ by one. One says that s has reached its *membership point*.

If both `CheckIa` and `CheckIb` fail, it is assumed that some transient disturbance has corrupted the frame of s'. Thus $m_s[s']$ is put to 0 and $CFail_s$ is increased by 1. The next station becomes the first successor of s.

When `CheckIa` fails but `CheckIb` passes, either s could be *send faulty* or s' could be *receive faulty*. According to the confidence principle, s assumes the latter, puts the membership of s' to 0 and increases $CFail_s$ by 1. However, s performs further similar checks over the frame received from the *next successor* s'' for double check:

`CheckIIa` CRC calculation with $m_s[s] = 1$ and $m_s[s'] = 0$.

`CheckIIb` CRC calculation with $m_s[s] = 0$ and $m_s[s'] = 1$.

If `CheckIIa` passes, s is confirmed that s' is *received faulty*. It increases $CAcc_s$ by 1 (for s'') and `CheckIIb` is discarded. Again, s has reached its *membership point*.

If `CheckIIb` passes, s assumes that it is itself *send faulty* and s' is non-faulty and leaves the active state.

If both `CheckIIa` and `CheckIIb` fail, s considers that s'' is faulty. Therefore it puts $m_s[s''] = 0$, it increases $CFail_s$ by 1 and will perform `CheckIIa` and `CheckIIb` again with the next successor following the same procedure.

It is assumed that at least 3 stations are active.

2.3 Clique avoidance mechanism

The *clique avoidance mechanism* reads as follows: Once per round, at the beginning of its time slot, a station s checks whether $CAcc_s > CFail_s$. If it is the case, it resets both counters as already said above and sends a message. If it is not the case, the station fails. It puts its own membership vector bit to 0, i.e., $m_s[s] = 0$, and leaves the `active` state, thus will not send in the subsequent rounds. The intuition behind this mechanism is that a station that fails to recognize a majority of frames as correct, is most probably not working properly. Other working stations, not receiving anything during the time slot of s, put the bit of s to 0 in their own membership vector.

It should be noted that implicit acknowledgment and clique avoidance mechanism interfere with each other, which contributes to make the analysis of the algorithm difficult.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

2.4 Re-integration

Faulty stations which have left the active state can re-integrate the active state (Kopetz 1999; Bauer and Paulitsch 2000). The re-integration algorithm that we describe here differs slightly from the one proposed in Kopetz and Grünsteidl (1999) to guarantee stabilisation after k faults. The algorithm proposed in Kopetz and Grünsteidl (1999) is enough to guarantee stabilisation after 1 fault, but not after k faults. A major difference is that re-integration, with our algorithm, may last longer, since a station, once it has acquired a membership vector, has to listen at least a full round before beeing able to send a frame, which is not necessarily the case with the algorithm proposed in Kopetz and Grünsteidl (1999).

A re-integrating station s copies the membership vector from some active station. As soon as the integrating station has a copy, it updates its membership vector listening to the traffic following the same algorithm as other working stations. During its first sending slot, it resets both counters, $CAcc_s$ and $CFail_s$ to 0, without sending any frame. During the following round, it increases its counters and keeps updating its membership vector as working stations do. At the beginning of its next sending slot, s checks whether $CAcc_s > CFail_s$. If it is the case, it puts $m_s[s]$ to 1 and sends a frame, otherwise it leaves the `active` state again. Receiving stations, if they detect a valid frame, put the membership of s to 1 and then perform the CRC checks as described above.

2.5 Example

Consider a set S composed of four stations and suppose that all stations received correct frames from each other for a while. This means that they all have identical membership vectors and $CFail = 0$. After station s_3 has sent, the membership vectors as well as counters $CAcc$ and $CFail$ look as follows. Remember that there is no global resetting of $CAcc$ and $CFail$. Resetting is relative to the position of the sending station.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	$CAcc$	$CFail$
s_0	1	1	1	1	4	0
s_1	1	1	1	1	3	0
s_2	1	1	1	1	2	0
s_3	1	1	1	1	1	0

We suppose that a fault occurs while s_0 is sending and that no subsequent fault occurs for at least two rounds, calculated from the time slot of s_0. We assume also that the frame sent by s_0 is recognized as correct by s_2 only. So the set S is split in two subsets, $S_1 = \{s_0, s_2\}$ and $S_0 = \{s_1, s_3\}$.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

1. Membership vectors and counters after s_0 has sent:

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	1	1	1	1	0
s_1	0	1	1	1	3	1
s_2	1	1	1	1	3	0
s_3	0	1	1	1	1	1

2. Membership vectors and counters after s_1 has sent. At this point `CheckIa` fails but `CheckIb` passes for s_0. However, `CheckIa` passes for s_3 (because of the fault, both `CheckIa` and `CheckIb` failed in the preceding time slot for s_3). Notice that s_2 does not have the same membership vector as s_1, so it does not recognize the frame sent by s_1 as correct.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	1	1	1	1
s_1	0	1	1	1	1	0
s_2	1	0	1	1	3	1
s_3	0	1	1	1	2	1

3. Membership vectors and counters after s_2 has sent. Now `CheckIIa` passes for s_0, but both `CheckIa` and `CheckIb` fail for s_1 because its membership vector differs with the one of s_2 on s_0. s_3 does not have the same membership vector as s_2, so it does not recognize the frame sent by s_2 as correct.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	1	1	2	1
s_1	0	1	0	1	1	1
s_2	1	0	1	1	1	0
s_3	0	1	0	1	2	2

4. Memberships and counters after the time slot of s_3, which cannot send due to the clique avoidance mechanism and leaves the `active` state:

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	1	0	2	1
s_1	0	1	0	0	1	1
s_2	1	0	1	0	1	0
s_3	0	0	0	0	0	0

5. Memberships and counters after s_0 has sent again. At this point `CheckIa` succeeds for s_2 while s_1 is still looking for a first successor.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	1	0	1	0
s_1	0	1	0	0	1	2
s_2	1	0	1	0	2	0
s_3	0	0	0	0	0	0

6. Memberships and counters after the time slot of s_1, which cannot send due the clique avoidance mechanism and leaves the `active` state:

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	1	0	1	0
s_1	0	0	0	0	0	0
s_2	1	0	1	0	2	0
s_3	0	0	0	0	0	0

Membership vectors are coherent again at this point of time.

3 Proving clique avoidance

In this section we prove that if k faults occur at a rate of more than 1 fault per two TDMA rounds and if no fault occur during two rounds following fault k, then at the end of that second round, all active stations have the same membership vector, so they form a single *clique* in the graph theoretical sense.

Let us denote by W the subset of S that contains all working stations. We may write $m_s = S'$ for a station s with $S' \subseteq S$ as a short hand for $m_s[s'] = 1$ iff $s' \in S'$. To prove coherence of membership vectors we start with the following situation. We suppose that stations of W have identical membership vectors and all have $CFail_s = 0$. Because $m_s[s] = 1$ for any working station, this implies that $m_s = W$ for any $s \in W$. Faults occur from this initial state.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

Let us define a graph as follows: the nodes are the stations, and there is an arc between s and s' iff $m_s[s'] = 1$. We recall that, in graph theory, a *clique* is a complete subgraph, i.e., each pair of nodes is related by an arc. Thus initially, W forms a single clique in the graph theoretical sense.

3.1 Introductory example

Let us illustrate how things might work in the case of two faults. The first fault occurs when s_0 sends. We suppose that only s_1 fails to receive correctly the frame sent by s_0. S is split as $S_1 = \{s_0, s_2, s_3\}$ and $S_0 = \{s_1\}$. Membership vectors and counters after s_0 has sent. At this point, `CheckIa` passes for s_3.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	1	1	1	1	0
s_1	0	1	1	1	3	1
s_2	1	1	1	1	3	0
s_3	1	1	1	1	2	0

Notice that $\{s_0, s_1, s_2, s_3\}$, do not form a clique anymore, the arc (s_1, s_0) is missing.

Membership vectors and counters after s_1 has sent. At this point, `CheckIa` fails but `CheckIb` passes for s_0. s_2 and s_3 do not have the same membership vector as s_1, so they don't accept its frame as correct.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	1	1	1	1
s_1	0	1	1	1	1	0
s_2	1	0	1	1	3	1
s_3	1	0	1	1	2	1

Membership vectors and counters after s_2 has sent. At this point, we suppose that a second fault occurs. Neither s_3 nor s_0 recognize the frame sent by s_2 as correct. S_1 is split in $S_{11} = \{s_2\}$ and $S_{10} = \{s_0, s_3\}$. At this point, s_0 keeps looking for a second successor and both `CheckIa` and `CheckIb` fail for s_1.

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	0	1	1	2
s_1	0	1	0	1	1	1
s_2	1	0	1	1	1	0
s_3	1	0	0	1	2	2

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

Membership vectors and counters after the time slot of s_3, which is prevented from sending because of the clique avoidance mechanism:

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	1	0	0	0	1	2
s_1	0	1	0	0	1	1
s_2	1	0	1	0	1	0
s_3	0	0	0	0	0	0

One notices that s_0, then s_1 are prevented from sending because of the clique avoidance mechanism. Membership vectors and counters after the time slot of s_1:

stations	$m[s_0]$	$m[s_1]$	$m[s_2]$	$m[s_3]$	*CAcc*	*CFail*
s_0	0	0	0	0	0	0
s_1	0	0	0	0	0	0
s_2	0	0	1	0	1	0
s_3	0	0	0	0	0	0

Coherence is achieved again after the time slot of s_1, where s_2 remains the only active station. Though S_{11} is smaller than S_{10}, the position of s_2 in the ring as the first station of the round with the second fault allows it to capitalize on frames accepted in the round before and to win over the set S_{10}.

3.2 Proving a single clique after k faults

The proof proceeds as follows. First we show a preliminary result. If W is divided into subsets S_i in such a way that all stations in a subset have the same membership vector, then stations inside a subset behave similarly: if there is no fault, they recognize the same frames as correct or as incorrect. Frames sent by stations from their own subset are the ones that they recognize as correct, while frames sent by stations from other subsets are all recognized as incorrect.

Then we show that the occurrence of faults does produce such a partitioning, i.e., after fault k, W is divided into subsets S_w, where $w \in \{0,1\}^k$. Indeed, as illustrated by the example above, after 1 fault, W is split in S_1, the stations that recognize the frame as correct, and S_0, the stations that do not recognize the frame as correct. Because any station recognizes itself as correct, S_1 is not empty. Now, suppose that a second fault occurs. Assume that the second fault occurs when a station from set S_1 sends. As before, set S_1 splits into S_{11}, the stations that recognize the frame as correct, and S_{10}, the stations that do not recognize the frame as correct. Again S_{11} is not empty. Set S_0 becomes S_{00} because stations in S_0 don't have the same membership vector as stations in S_1. And the process generalizes. If a station s from

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

a set S_w sends when fault k occurs, S_w splits into S_{w1} and S_{w0} with $S_{w1} \neq \emptyset$. Then, we show that two stations s and s' have the same membership vector if and only if they belong to the same set S_w. Using the preliminary lemma, we have a result about the incrementation of the counters *CAcc* and *CFail*, namely, all stations from a set S_w increment *CAcc* if a station from S_w sends, and increment *CFail* if a station from $S_{w'}$ sends, where $w \neq w'$. From this, we can deduce our main result: in the second round after fault k, only stations from a single set S_w can send. It follows that, at the end of that second round, there can be at most only one clique.

First we give a lemma that says that, if two stations have the same membership vector, then they recognize mutually their frames as correct.

Lemma 1
Let s and $s' \in W$ with $m_s = m_{s'}$. Then, $m_s[s'] = 1$ and $m_{s'}[s] = 1$.

Proof
$s, s' \in W$ means $m_s[s] = 1$ and $m_{s'}[s'] = 1$. Because s and s' have the same membership vector, one has $m_s[s'] = 1$ and $m_{s'}[s] = 1$. □

Now we give our preliminary result when active stations are divided into subsets S_i in such a way that all stations in a subset have the same membership vector.

Proposition 2
Suppose that W is divided into m subsets $S_1, \ldots, S_m$ such that s and s' have the same membership vector iff s and s' belong to the same subset S_i, $1 \leqslant i \leqslant m$. Let $s \in S_i$, $1 \leqslant i \leqslant m$. Assume that no fault occurs. Then, each time some other station s' is sending:

1. if $s' \in S_i$, s increases $CAcc_s$ by 1 and keeps the membership bit of s' to 1,
2. If $s' \in S_j$, $j \neq i$,
   (a) either s increases $CFail_s$ by 1 and puts the membership bit of s' to 0,
   (b) or s leaves the active state.

Proof
We prove this proposition for the general case where integrating stations are allowed. Let $s \in S_i$, or, if s is a station about to integrate, suppose it has copied its membership vector from an active station belonging to S_i.

First suppose $s' \in S_i$ or, if s' is an integrating station, s' has copied its membership vector from some active station $s'' \in S_i$. Following the integration policy, both s and s' put the membership bit of s' to 1 when s' sends. Because s and s' have the same membership vector and because no other fault occurs, s recognizes as correct the frame sent by s'. So it increases $CAcc_s$ by 1 when s' sends and keeps the membership bit of s' to 1, also if s is performing `CheckIa` or `CheckIIa`.

Suppose now $s' \in S_j$, $j \neq i$. As above, if s' is an integrating station, s' has copied its membership vector from some active station $s'' \in S_j$. Because s and s' do not have the same membership vector, s does not recognize the frame sent by s' as correct. If s has reached its membership point already or is a station about to integrate, it increases $CFail_s$ by 1 and puts the membership bit of s' to 0.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

If s has not reached its membership point yet, it performs either `checkI` or `CheckII`. Suppose first s performs `CheckIa` and `CheckIb`, i.e., s' could be the first successor of s. Because $s' \in S_j$, `CheckIa` does not pass. Hence, s increases $CFail_s$ by 1 and puts the membership bit of s' to 0.

Suppose now s performs `CheckIIa` and `CheckIIb`, i.e., s' could be the second successor of s. If s and s' disagree only on the bit for s, then `CheckIIb` fails for s which leaves the active state. Otherwise, s simply does not recognize as correct the frame sent by s'; it increases $CFail_s$ by 1, puts the membership bit of s' to 0 (and continues looking for a second successor). □

Now we prove a crucial proposition. Namely, occurence of faults divide the active stations into subsets characterized by their membership vectors.

Proposition 3
At the end of the time slot of s^k, the station where fault k occurs $k \geqslant 1$, W is partitioned into subsets S_w, with $w \in \{0,1\}^k$, such that two stations $s \in S_w$ and $s' \in S_{w'}$ have the same membership vector iff $w = w'$.

Proof
We proceed by induction on k.
Basis : $k = 1$. Let s^1 be the station which is sending when the first fault occurs. Before the time slot of s^1, all active stations, plus s^1 if s^1 is an integrating station, have the same membership vector, namely W, by hypothesis on the initial state. In case s^1 is an integrating station, all stations put the membership bit of s^1 to 1 by the integrating station policy. We denote by S_0 the subset of W that *failed* to receive correctly the frame sent by s^1 while we denote by S_1 the subset of stations that *accepted* the frame as correct.

All stations in S_1 could receive correctly the frame sent by s^1, consequently their membership vectors do not change. None of the station in S_0 could receive correctly the frame sent by s^1 because of the fault. Hence they all put the membership bit of s^1 to *false*, also in the case where s^1 was an integrating station. Thus they all have the same membership vector, namely $W \setminus \{s^1\}$.

Suppose the result is true till fault $k-1$.
Induction step: we prove that the result is true for k, $k > 1$. Let s^k be the station which is sending when fault k occurs. By induction hypothesis, at the end of the time slot of s^{k-1}, W is partitioned into sets S_w with $w \in \{0|1\}^{k-1}$. There is a unique $w \in \{0|1\}^{k-1}$ with $s^k \in S_w$ or such that the membership vector of s^k is identical to stations of S_w, in case s^k is an integrating station.

First, we show that S_w splits into S_{w0} and S_{w1}. By Lemma 1, $m_s[s^k] = 1$ for any station $s \in S_w$. Let us denote by S_{w1} the subset of S_w that could receive s^k correctly and by S_{w0} the subset of S_w that could not receive s^k correctly. Obviously, S_{w1} and S_{w0} partition S_w. After the time slot of s^k, all stations in S_{w1} keep the membership of s^k to 1, while all stations in S_{w0} put it to 0. Thus $s, s' \in S_w$ have the same membership vector if and only if $s, s' \in S_{w1}$ or $s, s' \in S_{w0}$.

Consider some $w' \in \{0|1\}^{k-1}$, with $w' \neq w$ and $S_{w'} \neq \emptyset$. Let $s' \in S_{w'}$. We show that $S_{w'}$ becomes $S_{w'0}$ and that the condition on the membership vectors still holds.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

$w' \neq w$ means that $m_{s'}$ and m_{s^k} differ on some bit s^*. So, s' cannot recognize as correct the frame sent by s^k and $S_{w'}$ can be denoted by $S_{w'0}$.

Obviously, $\forall s \in S_{w1} : m_{s'} \neq m_s$. Could it be that now stations $S_{w'0}$ and S_{w0} have the same membership vector? This could be only if their membership vectors differed only on the bit for s^k. But this would mean that station s^k has already emitted, that a fault occured and station in $S_{w'}$ did not accept the frame as correct while stations in S_w did. Because all stations emit in one round, some station u from $S_{w'}$ has emitted. By Proposition 2, membership vectors of stations in $S_{w'}$ and S_w differ on u and $u \neq s^k$. Hence, stations $S_{w'0}$ and S_{w0} have different memberships vectors.

Using a similar argument, membership vectors of stations in $S_{w''0}$ and $S_{w'0}$ remain different with $w'' \in \{0|1\}^{k-1}$, $w' \neq w'' \neq w$ and $S_{w''} \neq \emptyset$. □

Finally, we show that only stations from a unique set S_w are able to send in the second round following the first fault.

Theorem 4

Suppose some station is able to send in the second round following fault k. Let us denote this station by s. By Proposition 3, exists some $w \in \{0|1\}^k$ such that $s \in S_w$. Then, only stations from S_w can send in the second round following fault k.

Proof

Let $CAcc_s$ and $CFail_s$ when s performs the clique avoidance mechanism. By Proposition 2, $CAcc_s = |\{s' \in S_w \text{ s. t. } s' \text{ sent in the first round following fault } k\}|$ and $CFail_s = \Sigma_{w' \neq w} |\{s' \in S_{w'} \text{ s. t. } s < s', \text{ and } s' \text{ sent in the first round following fault } k\}|$, where $<$ refers to the statical order among stations. Because s is able to send, one has $CAcc_s > CFail_s$ at the beginning of the time slot of s in the second round.

Let t be the follower station of s in the statical order ready to send after s. Is t able to send, or is it prevented from sending by the clique avoidance mechanism? We show that t is able to send if and only if it belongs to S_w.

Suppose $t \in S_w$. When its time slot comes in the second round, it has increased $CAcc_t$ by 1 when s has sent in the second round. Because, in the first round following fault k, t increases its counters as s does by Proposition 2, one has $CAcc_t = CAcc_s$, or $CAcc_t = CAcc_s + 1$ in case s was an integrating station, and $CFail_t = CFail_s$ at the beginning of the time slot of t in the second round. Thus $CAcc_t > CFail_t$ and t is able to send as well.

Suppose now $t \in S_{w'}$ for some $w' \neq w$. $CAcc_t = |\{s' \in S_{w'} \mid t \leqslant s' \leqslant s^k \text{ and } s' \text{ sent in the first round following fault } k\}|$. Indeed, between s^k and its present time slot, t has not accepted any frame since only s has sent. However, by Proposition 2, $CAcc_t \leqslant CFail_s$ since all frames accepted by t are not recognized as correct by s. For a similar reason, $CAcc_s \leqslant CFail_t$. It follows that, at the beginning of its time slot, $CAcc_t < CFail_t$ and t is prevented from sending.

A similar argument can be repeated to all stations ready to send in the second round giving the result. □

From Theorem 4, one deduces that, at the end of the second round following fault k, for any station $s \in S_w$: $m_s = S_w$. Using Lemma 1, this gives our safety property about cliques.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

Corollary 5
At the end of the second round following fault 1, all working stations form a single clique in the graph theoretical sense.

4 Automatic verification: the 1 fault case

In the case of a single fault, the set W of active stations is divided into two subsets, S_1 and S_0. The set S_1 is not empty as it contains s^1, the station that was sending when the fault occurs. We assume that no other fault occurs for the next two rounds, a round is taken with the beginning of the time slot of s^1. We want to prove automatically for an arbitrary number N of stations that, at the end of the second round following the fault, all working stations form a single non-empty clique. To achieve this goal, we need a formalism to model the protocol and a formalism to specify the properties that the protocol must satisfy. To model the protocol, we take synchronous automata extended with parameters and counters. To specify the properties, we take the temporal logic CTL. To keep the number of parameters as low as possible, we do not consider re-integrating stations.

To be able to verify automatically the protocol for the parametric case where the number of stations is a paramater N, we need to abstract the behavioural model of the N identical extended automata into a single extended automaton. The abstraction we use is the standard (infinite) counter abstraction and can be automatized. The novelty and difficulty of our case study, compared with other examples using a similar abstraction technique (Pnueli *et al.* 2002; Delzanno 2000), lies in the fact that each individual extended automaton that models one station has local infinite variables: two counters, *CAcc* and *CFail*, and a membership vector m that all depends on the paramater N. Applying counter abstraction directly would lead to a too coarse model, useless for verification. Consequently, before we apply the abstraction, we perform a transformation of the N extended automata in order to replace local variables in guards by a finite number of global counters. The successive models we obtain are related by a (bi)simulation property.

We divide the presentation in four main parts: first, we draw straightforward consequences from section 3 for the 1 fault case. Second, we give the behavioural model of the protocol under the form of N synchronous automata with local variables. Third we show how we build the abstract model replacing local variables by a finite number of global counters, strenthening guards and performing the usual counter abstraction in such a way that each successive model (bi)simulates the previous one. Finally, we give the properties that have been automatically proved on the resulting model which consists of a single extended automaton. This establishes the 'non-clique' property.

4.1 Properties of the 1-fault case

We shall make use of the results presented in this section in the stepwise transformations where local variables are replaced by a finite number of global counters.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

Proposition 6
Let s be the sending station when the fault occurs. In the round following the fault, the 3 conditions below are equivalent:

1. s leaves the active state after `CheckI` and `CheckII`,
2. the two follower stations of s have 1 everywhere in their membership vector, except for the bit for s which is 0,
3. the two follower stations of s are in S_0.

Proof
A simple consequence of the fact that before the fault, all membership vectors are all equal with 1 in each bit. □

Proposition 7
Let s be the sending station when the fault occurs and suppose that no fault occurs during the two subsequent rounds. Then, only station s can leave the active state because of `CheckI` and `CheckII` in the round following the fault. In later rounds, `CheckI` and `CheckII` do not play any rôle.

Proof
If s has left because of `CheckI` and `CheckII`, there is nothing to prove. Otherwise, the first or second successor of s belongs to S_1, so either `CheckIa` or `CheckIIa` succeeds in this round or in later rounds. Consider now $s' \neq s$. Could s' leave the active state because of `CheckI` and `CheckII` ? Consider s'' the station sending after s'. Suppose that `CheckIa` fails (otherwise there is nothing to prove). This means that s' and s'' have different membership vectors. Because no new fault occurs, the difference in the membership vector can not be at the bit for s' only. Thus s' discards both `CheckIa` and `CheckIb` and keeps looking for a first successor. □

The result below shows that we can replace the N individual counters $CAcc$ and $CFail$ by two global counters $d0$ and $d1$. Let $d1$ be a counter to count how many stations of S_1 have sent so far in the round since the fault occurred. Let $d0$ be a similar counter for S_0. Let s be a station ready to send. Assume $s \in S_1$. How much is $CFail_s$? It is exactly given by $d0$. How much is $CAcc_s$? Generally, it is more than $d1$. One has to add all stations that have emitted before the fault since the last time slot of s, because s has recognized them all as correct, see Figure 2. However, this number can be calculated exactly with the help of $d1$ and $d0$ only as Theorem 8 shows.

Theorem 8
Let s a station ready to send and suppose there is only 1 fault.
In the round following fault 1:

1. If $s \in S_1$, then $CAcc_s = \mid W \mid - d0$ and $CFail_s = d0$.
2. If $s \in S_0$, then $CAcc_s = \mid W \mid - d1$ and $CFail_s = d1$.

In later rounds:

1. If $s \in S_1$, then $CAcc_s = \mid S_1 \mid$ and $CFail_s = \mid S_0 \mid$.
2. If $s \in S_0$, then $CAcc_s = \mid S_0 \mid$ and $CFail_s = \mid S_1 \mid$.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

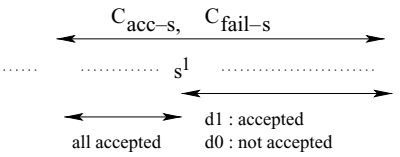


Fig. 2. Illustrating proof of Theorem 8.

Proof

First round after the fault. We prove the first item only, the second one being dual. Since there was no fault in the last round, s has recognized as correct all stations between itself and s^1 in the last round. By Proposition 2, it has recognized as correct all stations of S_1 that have sent so far in the round. This is illustrated in Figure 2. Thus $CAcc_s =\mid \{s' \mid s \leqslant s' < s^1\} \cap W \mid +d1$, where $\leqslant$ and $<$ refer the statical order among stations. $\{s' \mid s \leqslant s' < s^1\} \cap W = W \setminus (\{s' \mid s^1 \leqslant s' < s\} \cap W)$. Thus $\mid \{s' \mid s \leqslant s' < s^1\} \cap W \mid =\mid W \mid -d1 - d0$. This gives $CAcc_s =\mid W \mid -d0$. By Proposition 2, $CFail_s = d0$.

Second and later rounds after the fault. Let $s \in S_1$ be the station whose time slot comes first in the second round. This station is the faulty station itself. By Proposition 2, $CAcc_s$ is the number of stations of S_1 that could send in the previous round, i.e., that are still active, which is precisely $\mid S_1 \mid$. Similarly, $CFail_s =\mid S_0 \mid$. If s can send, $\mid S_1 \mid$ remains unchanged, if s cannot send, $\mid S_1 \mid$ diminishes by 1. At the beginning of the follower time slot, $\mid S_1 \mid$ is exactly the number of stations from S_1 that have send in the round preceding this time slot, and similarly for S_0. Let us call s' the station corresponding to that time slot. If $s' \in S_1$, using Propositions 2, $CAcc_{s'} =\mid S_1 \mid$ and $CFail_s =\mid S_0 \mid$. If $s' \in S_0$, the dual is true. If $s' \notin S_0 \cup S_1$, s' has not send in the preceding round. In any case, at the end of the time slot of s', $\mid S_1 \mid$ is exactly the number of stations from S_1 that have send in the round, and similarly for S_0. Using Proposition 2 and repeating the same argument for all time slots that follow gives the result. □

4.2 Behavioural model

We give in this section a formal description of the TTP membership algorithm. For that, we consider a general specification formalism for parametrized networks. We assume that systems can have an arbitrary number N of components which may share global variables and can have also local variables. Moreover, these components can communicate by broadcasting messages. To describe the behaviour of each of the components, we adopt an extended automata-based formalism (described using the notations of Manna and Pnueli (1995)) where each transition between control state is a guarded command which may involve (broadcast) communications. We assume that the executions of all the parallel components are synchronous following the semantical model of Benveniste and Berry (1991) (i.e., all components have the same speed w.r.t. a global logical clock defining a notion of execution step in the system, and at each step, all the operations, including broadcast communications, are instantaneous).

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

The formal semantics of such models can be defined in terms of a transition system. A state defines a global configuration of the network corresponding to the values of the global variables together with the values of all local variables (including the control states) for each of the N components of the network. Transitions between states are defined straightforwardly according to the semantical model mentioned above. The behaviours of the network are defined as the possible execution paths in the so defined transition system.

Before giving the formal description of the TTP, let us introduce some notations. We denote the sending of a broadcast message a by $a!!$ and the receiving of a broadcast message a by $a??$. Otherwise, [] denotes non-deterministic choice, $t^{\oplus\oplus}$ stands for $t := t \oplus 1$, i.e., the value of t in the next state is incremented by 1 modulo N, A^{++} stands for $A := A + 1$, and ';' denotes sequentiality. Assignments separated by ',' on the right side of $\longrightarrow$ could happen in any order. Variables not mentioned in transitions remain unchanged.

Figure 3 gives the formal specification of the TTP membership algorithm.

The protocol is composed of two inputs and N processes $P[i]$ running in parallel. The inputs of the protocol are the parameter N and a *boolean fault* which is *true* in case a fault occurs. Each station P is a non-deterministic finite state machine extended with local variables. The local variables are the counters A and F for *CAcc* and *CFail*, the membership vector m, the variable t to keep track of the time slots, and the variable s to remember the identity of the station which is sending when a fault occurs. Following Manna and Pnueli (1995), all local variables are marked with the identity of the station they belong to, which is denoted by $[i]$.

The parallel composition is synchronous and the automata synchronize on the broadcast message *emit*.

There are four locations l_{in}, l_0, l_1 and l_F. Location l_{in} is the initial location or state. When a fault occurs, stations that recognize the fault move to state l_0 while stations that do not recognize the fault move to l_1. Stations who are prevented from sending move to location l_F.

Let us go through all transitions of location l_{in} in details. In the initial state l_{in}, all stations have the same membership vectors. First, suppose that there is no fault. A station can always emit when it is its turn to do so, expressed in the model by the condition $t[i] = i$. The first transition has $t[i] = i \wedge \neg fault$ as guard. The action performed by that transition is: $emit!!, t[i]^{\oplus\oplus}, A[i] := 1;\ l_{in}$. Put informally, when it is its turn to emit and there is no fault, a station sends a frame, increments the variable t by 1 modulo N, resets the counter A to 1 and, then, remains in state l_{in}. The third transition models the behaviour of a receiving station when there is no fault. In that case, it always recognizes as correct frames sent by other stations. This is expressed in the model by the guard $t[i] \neq i \wedge emit?? \wedge \neg fault$. These two guards, from the first and third transition, for two different processes i and j are true simultaneously. Indeed, in the synchronous model of computation, a computational step 'takes no time' (Benveniste and Berry 1991), therefore the emission of *emit* by process i is synchronous with its reception by all other processes. When the parallel statement is finished, all stations have incremented t by 1, the emitting station has reset A to 1 while other stations have incremented A. Counter F stays at its

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

in N : int,$N > 1$ in $fault$: boolean signal

$\|_{i=1}^{N} P[i]$, where $P[i]$ is the following program:

$A[i]$: int, init $N+1-i$ $F[i]$: int, init 0
$m[i]$: array of boolean, init 1 $t[i]$: int, init 1 // t for turn
$s[i]$: int, init 0// identity of the faulty station

loop forever do

l_{in}
$[]\ t[i] = i \wedge \neg fault \longrightarrow emit!!, t[i]^{\oplus\oplus}, A[i] := 1;\ l_{in}$
$[]\ t[i] = i \wedge fault \longrightarrow emit!!, t[i]^{\oplus\oplus}, A[i] := 1, s[i] := t[i];\ l_1$
$[]\ t[i] \neq i \wedge emit?? \wedge \neg fault \longrightarrow t[i]^{\oplus\oplus}, A[i]^{++};\ l_{in}$
$[]\ t[i] \neq i \wedge emit?? \wedge fault \longrightarrow t[i]^{\oplus\oplus}, A[i]^{++}, s[i] := t[i];\ l_1$
$[]\ t[i] \neq i \wedge emit?? \wedge fault \longrightarrow t[i]^{\oplus\oplus}, F[i]^{++}, s[i] := t[i], m[i][t[i]] := 0;\ l_0$

l_0
$[]\ t[i] = i \wedge A[i] > F[i] \longrightarrow emit!!, t[i]^{\oplus\oplus}, A[i] := 1, F[i] := 0;\ l_0$
$[]\ t[i] = i \wedge A[i] \leqslant F[i] \longrightarrow \neg emit!!, t[i]^{\oplus\oplus};\ l_F$
$[]\ t[i] \neq i \wedge emit?? \wedge m[t[i]] = m[i] \longrightarrow t[i]^{\oplus\oplus}, A[i]^{++};\ l_0$
$[]\ t[i] \neq i \wedge emit?? \wedge m[t[i]] \neq m[i] \longrightarrow t[i]^{\oplus\oplus}, F[i]^{++}, m[i][t[i]] := 0;\ l_0$
$[]\ t[i] \neq i \wedge \neg emit?? \longrightarrow t[i]^{\oplus\oplus}, m[i][t[i]] := 0;\ l_0$

l_1
$[]\ t[i] = i \wedge A[i] > F[i] \longrightarrow emit!!, t[i]^{\oplus\oplus}, A[i] := 1, F[i] := 0;\ l_1$
$[]\ t[i] = i \wedge A[i] \leqslant F[i] \longrightarrow \neg emit!!, t[i]^{\oplus\oplus};\ l_F$
$[]\ t[i] \neq i \wedge emit?? \wedge i = s[i] \wedge t[i] = s[i] + 2 \wedge (m[t[i]][s[i]] = m[t[i]-1][s[i]] = 0) \longrightarrow t[i]^{\oplus\oplus};\ l_F$
$[]\ t[i] \neq i \wedge emit?? \wedge i = s[i] \wedge t[i] = s[i] + 2 \wedge \neg(m[t[i]][s[i]] = m[t[i]-1][s[i]] = 0) \longrightarrow t[i]^{\oplus\oplus}, A[i]^{++};\ l_1$
$[]\ t[i] \neq i \wedge emit?? \wedge (i \neq s[i] \vee t[i] \neq s[i] + 2) \wedge m[t[i]] = m[i] \longrightarrow t[i]^{\oplus\oplus}, A[i]^{++};\ l_1$
$[]\ t[i] \neq i \wedge emit?? \wedge (i \neq s[i] \vee t[i] \neq s[i] + 2) \wedge m[t[i]] \neq m[i] \longrightarrow t[i]^{\oplus\oplus}, F[i]^{++}, m[i][t[i]] := 0;\ l_1$
$[]\ t[i] \neq i \wedge \neg emit?? \longrightarrow t[i]^{\oplus\oplus}, m[i][t[i]] := 0;\ l_1$

l_F
$[]\ t[i] = i \longrightarrow \neg emit!!, t[i]^{\oplus\oplus};\ l_F$
$[]\ t[i] \neq i \longrightarrow t[i]^{\oplus\oplus};\ l_F$

end loop

Fig. 3. The algorithm with N stations and 1 fault, $M1$.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

initial value 0 since it is not incremented in transitions containing the condition $\neg fault$.

Suppose now that there is a fault, which is modeled by the condition $fault$. For an emitting station, this is the second transition. The action taken by the emitting station is similar to the first transition, except that it records its identity with $s[i] := t[i]$ and then moves to location l_1 since, by the confidence principle, a station never thinks of itself as faulty (rather receiving stations are faulty). For a receiving station the guard $t[i] \neq i \wedge emit?? \wedge fault$ is true. The occurrence of an asymmetric fault is modeled by a non-deterministic choice represented by the fourth and fifth transitions. The information of whether a station recognizes the fault is recorded in the control via locations l_0 and l_1. If a receiving station recognizes the fault, it increases F by 1, $F[i]^{++}$, memorizes the identity of the faulty station, $s[i] := t$, puts the membership bit of the sending station to 0, $m[i][t[i]] := 0$ and moves to l_0 (fifth transition). If a station does not recognize the fault, it increases A by 1, $A[i]^{++}$, and moves to l_1.

In location l_0, a station behaves as follows. In its time slot, $t[i] = i$, either it passes the clique avoidance mechanism, $A[i] > F[i]$ (first transition), and emit, $emit!!$, reset its counters, $A[i] := 1, F[i] := 0$, increments the time slot, $t[i]^{\oplus\oplus}$, and stays in l_0, or the clique avoidance mechanism fails $A[i] \leqslant F[i]$ (second transition), it cannot emit, $\neg emit!!$, and goes to the fail state l_F. Outside its time slot, $t[i] \neq i$, if a message has been broadcast, $emit??$, and it has the same membership vector as the sending station, $m[t[i]] = m[i]$ (third transition), it increases the counter of accepted messages, $A[i]^{++}$, increments the time slot, $t[i]^{\oplus\oplus}$, and stays in l_0; if it does not have the same membership vector as the sending station, $m[t[i]] \neq m[i]$ (fourth transition), it increases the counter of failed messages, $F[i]^{++}$, puts the membership bit of the sending station to 0, $m[i][t[i]] := 0$, increments the time slot, $t[i]^{\oplus\oplus}$, and stays in l_0. Finally, if no message has been broadcast, $\neg emit??$ (fifth transition), it puts the membership bit of the station that failed sending to 0, $m[i][t[i]] := 0$, and stays in l_0.

In location l_1, transitions are similar except that two more transitions are added to cover the result of `CheckI` and `CheckII` using Propositions 6 and 7. These are the third and the fourth transitions. If the sending station is the second successor of the faulty station, $t[i] = s[i] + 2$, and the two successors of the faulty station have the membership bit of $s[i]$ to 0, $m[t[i]]s[i] = m[t[i] - 1]s[i] = 0$, then station $s[i]$ moves to l_F, otherwise station $s[i]$ stays in l_1.

In location l_F, stations keep only track of the time slot, they cannot send and stay there.

4.3 Construction of an abstract model

We show in this section the construction of a counter automaton which is an abstract model of the parametrized membership algorithm for an arbitrary number of components N. To simplify the presentation and the proof of the abstraction, we present this construction in several basic steps. The aim of the first steps is to encode the infinite-data-domain local variables of the N components with a finite number of global variables (counters). Then, the last step is a counter abstraction

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

which encodes the control configurations (for N components) with global variables counting the number of components at each control location.

In the sequel, we give the different abstraction steps by defining each time the abstract model and by showing that it (bi)simulates the previous one.

4.3.1 Eliminating identical locals t and s

The first transformation we perform is to replace N local variables $t[i]$ and $s[i]$ by two global counters t_G and s_G. It relies on the following fact:

Lemma 4.1
The two following properties are invariants of the program $M1$:

1. $\forall i, j \in \{1 \dots N\} : (t[i] = t[j] \ \wedge \ s[i] = s[j])$
2. $\exists ! i \in \{1 \dots N\} : t[i] = i$

In other words, at any computation step, all processes $P[i]$ have identical values for locals $s[i]$ and $t[i]$ and there is exactly one process whose identity is $t[i]$. We define a program $M2$ where these local variables are replaced by t_G and s_G in the following manner. We modify the transitions in order to encode the updates of the local variables as updates of the global ones. Since all processes are synchronous, these updates must be done by exactly one component. We choose that this will be done by the component for whom it is the turn to emit, i.e., who satisfies $t[i] = i$.

So, the program $M2$ is obtained from $M1$ by applying the following transformations: (1) initialize t_G and s_G to 1 and 0 respectively, (2) replace each occurrence of $t[i]$ in the guards by t_G, and (3) in all the guarded commands, if $t[i] = i$ appears in the guard, then replace $t[i]$ by t_G (resp. $s[i]$ by s_G), else remove the update statements of $t[i]$ and $s[i]$.

For example the second transition at location l_{in} (see Figure 3) is transformed into

$$t_G = i \ \wedge \ fault \ \longrightarrow \ emit!!, t_G^{\oplus\oplus}, A[i] := 1, s_G := t_G; \ l_1$$

We establish now that $M2$ bisimulates $M1$. For that, let us consider the relation $\alpha_{1,2}$ between states of $M1$ and $M2$ such that, for every σ_1 a state of $M1$, and for every σ_2 a state of $M2$, we have $\alpha_{1,2}(\sigma_1, \sigma_2)$ if and only if (1) σ_2 and σ_1 coincide on all locations and variables different from the locals t and s, and the globals t_G and s_G, and (2) $\sigma_2(t_G) = \sigma_1(t[i])$ and $\sigma_2(s_G) = \sigma_1(s[i])$ for every $i \in \{1, \dots, N\}$, i.e., the value of t_G and s_G in σ_2 is the same as the value of $t[i]$, respectively $s[i]$ in σ_1. Then, it can easily be checked that the relation $\alpha_{1,2}$ is a bisimulation between the transition systems of $M1$ and $M2$.

4.3.2 Eliminating locals A and F

Based on Theorem 8, we define a new model where local variables A and F are simulated by global counters d_0, d_1, C_0, and C_1; the counters C_0 and C_1 stand for $|S_0|$ and $|S_1|$ respectively. We need in addition a variable r to count the current round and a variable C_p which counts the number of steps performed in the current round.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

We define hereafter a program $M3$ obtained from $M2$ by the following transformations: (1) transitions starting at location l_{in} where the fault is detected ($fault$ appears in the guard), replace $A[i]$ and $F[i]$ by C_1 and C_0 respectively; (2) each transition starting from locations l_0 and l_1 which corresponds to an emit action (where $t_G = i$ appears in the guard) is duplicated into three transition corresponding to the cases where the execution is inside the first round, is inside some later round, or is precisely at the beginning of a new round. The comparisons of A and F in the guards and their updates are replaced by corresponding comparisons and updates on C_0, C_1, d_0, and d_1, according to Theorem 8; and (3) all other statements involving A and F are removed.

For example the fourth transition at location l_{in} :

$$t_G \neq i \ \wedge \ emit?? \ \wedge \ fault \quad \longrightarrow \ t_G^{\oplus\oplus}, A[i] := 1; \ l_1$$

is transformed into

$$t_G \neq i \ \wedge \ emit?? \ \wedge \ fault \quad \longrightarrow \ t_G^{\oplus\oplus}, {C_1}^{\oplus\oplus}; \ l_1.$$

The second transition at location l_0 :

$$t_G = i \ \wedge \ A[i] \leqslant F[i] \ \longrightarrow \ \neg emit!!, t_G^{\oplus\oplus}; l_F$$

is duplicated into three transitions t_1, t_2, t_3 as follows:

$$\begin{array}{c} t_G = i \ \wedge \ C_0 + C_1 \leqslant 2 \times d_1 \ \wedge \ C_p < N \ \wedge \ r = 0 \ \longrightarrow \ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_0^{--}; l_F \\ t_G = i \ \wedge \ C_0 \leqslant C_1 \ \wedge \ C_p = N \ \longrightarrow \ \neg emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}, C_0^{--}; l_F. \\ t_G = i \ \wedge \ C_0 \leqslant C_1 \ \wedge \ C_p < N \ \wedge \ r > 0 \ \longrightarrow \ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_0^{--}; l_F \end{array}$$

The obtained program $M3$ is bisimilar to $M2$ by Theorem 8.

4.3.3 Eliminating the local m

At this stage, we can see that the information given by m is not relevant anymore except in the case where a faulty station must leave the active state because its first and second successors have recognized it as faulty. To deal with this case, we replace the test on the vector m by the faulty station with a guess of the diagnostic of its two immediate successors (`checkI` and `checkII`). This guess is made exactly one round after the fault. For that, we simply perform a nondeterministic choice whether to leave or to stay in the active state. It turns out that this abstraction is precise enough for our purpose.

Then, we define a new program $M4$ obtained from the program $M3$ by (1) removing the transitions starting from l_1 involving tests on the membership bit vectors of the two immediate successors, (2) duplicating the transitions from l_1 for the case $(C_p = N) \wedge (r = 0)$ into two transitions corresponding to the actions of leaving or staying in the active state with g to fix the choice, and (3) removing all the remaining statements involving m.

Moreover, after the transformation above, guarded commands starting at l_0 and l_1 which involve the test $t_G \neq i$ can actually be compacted into trivial self loops. This leads to the program $M4$ given in Figure 4. One shows that this program simulates the program $M3$ by induction.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

in N	:	int, $N > 1$	C_0	:	int, init 0	t_G	:	int, init 1	r	:	int, init 0
in $fault$	:	bool. signal	C_1	:	int, init 1	s_G	:	int, init 0	d_0	:	int, init 0
in g	:	bool. signal	C_p	:	int, init 1	d_1	:	int, init 1			

$\|_{i=1}^{N} P[i]$, where $P[i]$ is the following program:

loop forever do

l_{in}
$$\begin{cases}
[] \; t_G = i \wedge \neg fault \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}; \; l_{in} \\
[] \; t_G = i \wedge fault \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, s_G := t_G; \; l_1 \\
[] \; t_G \neq i \wedge emit?? \wedge \neg fault \\
\quad \longrightarrow \; ; \; l_{in} \\
[] \; t_G \neq i \wedge emit?? \wedge fault \\
\quad \longrightarrow \; C_1^{++}; \; l_1 \\
[] \; t_G \neq i \wedge emit?? \wedge fault \quad \longrightarrow \; C_0^{++}; \; l_0
\end{cases}$$

l_0
$$\begin{cases}
[] \; t_G = i \wedge C_0 + C_1 > 2 \times d_1 \wedge C_p < N \wedge r = 0 \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_0^{++}; \; l_0 \\
[] \; t_G = i \wedge C_0 > C_1 \wedge C_p = N \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}; \; l_0 \\
[] \; t_G = i \wedge C_0 > C_1 \wedge C_p < N \wedge r > 0 \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p^{++}; \; l_0 \\
[] \; t_G = i \wedge C_0 + C_1 \leqslant 2 \times d_1 \wedge C_p < N \wedge r = 0 \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_0^{--}; \; l_F \\
[] \; t_G = i \wedge C_0 \leqslant C_1 \wedge C_p = N \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}, C_0^{--}; \; l_F \\
[] \; t_G = i \wedge C_0 \leqslant C_1 \wedge C_p < N \wedge r > 0 \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_0^{--}; \; l_F \\
[] \; t_G \neq i \quad \longrightarrow \; ; \; l_0
\end{cases}$$

l_1
$$\begin{cases}
[] \; t_G = i \wedge C_0 + C_1 > 2 \times d_0 \wedge C_p < N \wedge r = 0 \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_1^{++}; \; l_1 \\
[] \; t_G = i \wedge C_1 > C_0 \wedge C_p = N \wedge r \neq 0 \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}; \; l_1 \\
[] \; t_G = i \wedge C_1 > C_0 \wedge C_p = N \wedge r = 0 \wedge \neg g \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}; \; l_1 \\
[] \; t_G = i \wedge C_1 > C_0 \wedge C_p = N \wedge r = 0 \wedge g \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}, C_1^{--}; \; l_F \\
[] \; t_G = i \wedge C_1 > C_0 \wedge C_p < N \wedge r > 0 \\
\quad \longrightarrow \; emit!!, t_G^{\oplus\oplus}, C_p^{++}; \; l_1 \\
[] \; t_G = i \wedge C_0 + C_1 \leqslant 2 \times d_0 \wedge C_p < N \wedge r = 0 \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_1^{--}; \; l_F \\
[] \; t_G = i \wedge C_1 \leqslant C_0 \wedge C_p = N \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++}, C_1^{--}; \; l_F \\
[] \; t_G = i \wedge C_1 \leqslant C_0 \wedge C_p < N \wedge r > 0 \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_1^{--}; \; l_F \\
[] \; t_G \neq i \quad \longrightarrow \; ; \; l_1
\end{cases}$$

l_F
$$\begin{cases}
[] \; t_G = i \wedge C_p < N \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}; \; l_F \\
[] \; t_G = i \wedge C_p = N \\
\quad \longrightarrow \; \neg emit!!, t_G^{\oplus\oplus}, C_p' := 1, r^{++}; \; l_F \\
[] \; t_G \neq i \quad \longrightarrow \; ; \; l_F
\end{cases}$$

end loop

Fig. 4. Eliminating all locals, $M4$.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

4.3.4 Strengthening guards

Before the counter abstraction step where identities of processes will be lost, we need to strengthen the guards using some invariants of the system.

Lemma 4.2
The following statements are invariants of the program $M4$:

1. $\forall i\,(l_0[i] \wedge t_G = i \wedge C_0 + C_1 > 2 \times d_1 \wedge C_p < N \wedge r = 0) \Rightarrow d_0 < C_0)$
2. $\forall i\,(l_0[i] \wedge t_G = i \wedge C_0 + C_1 \leqslant 2 \times d_1 \wedge C_p < N \wedge r = 0) \Rightarrow d_0 < C_0)$
3. $\forall i\,(l_1[i] \wedge t_G = i \wedge C_0 + C_1 > 2 \times d_0 \wedge C_p < N \wedge r = 0) \Rightarrow d_1 < C_1)$
4. $\forall i\,(l_1[i] \wedge t_G = i \wedge C_0 + C_1 \leqslant 2 \times d_0 \wedge C_p < N \wedge r = 0) \Rightarrow d_1 < C_1)$

Invariant (1) in the lemma above says that when process $P[i]$ is at location l_0, if it is the turn of this process to emit, and if it is allowed to emit in the round following the fault, then not all stations from the set S_0 have emitted in that round. This is due to the fact that d_0 counts stations from the set S_0 that have emitted in that current round.

Thus, at location l_0, the guard $t_G = i \wedge C_0 + C_1 > 2 \times d_1 \wedge C_p < N \wedge r = 0$ can be strengthened into $t_G = i \wedge C_0 + C_1 > 2 \times d_1 \wedge C_p < N \wedge r = 0 \wedge d_0 < C_0$ without changing the semantics of the program. We do similar transformation using the other invariants.

Further, we update d_0 and d_1 in other transitions at l_0 and l_1 so that they keep counting the number of stations from the set S_0, S_1 respectively, that have emitted in subsequent rounds. In that way, we obtain more invariants similar to the ones given in Lemma 4.2 and we use them to strengthen guards without changing the semantics of the program.

For example, at l_0 transition

$$t_G = i \wedge C_0 > C_1 \wedge C_p = N \longrightarrow emit!!, t_G^{\oplus\oplus}, C_p := 1, r^{++};\ l_0$$

is changed into

$$t_G = i \wedge C_0 > C_1 \wedge C_p = N \longrightarrow emit!!, t_G^{\oplus\oplus}, d_0 := 0, C_p := 1, r^{++};\ l_0$$

and transition

$$t_G = i \wedge C_0 > C_1 \wedge C_p < N \wedge r > 0 \longrightarrow emit!!, t_G^{\oplus\oplus}, C_p^{++};\ l_0$$

is changed into

$$t_G = i \wedge C_0 > C_1 \wedge C_p < N \wedge r > 0 \longrightarrow emit!!, t_G^{\oplus\oplus}, d_0^{++}, C_p^{++};\ l_0.$$

It can be shown that

$$\forall i\,(l_0[i] \wedge t_G = i \wedge C_0 > C_1 \wedge C_p = N) \Rightarrow d_0 = C_0)$$

is an invariant and therefore the guard can be strengthened into

$$t_G = i \wedge C_0 > C_1 \wedge C_p = N \wedge d_0 = C_0.$$

Similarly to transitions starting from l_0 and l_1 we need to strengthen the guards of the transitions starting from l_F. For that, we introduce two supplementary counters

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

d_F and C_F – initial value 0 – which play a similar rôle for location l_F as counters d_0 and C_0 do for location l_0. We use invariants similar to those given in the lemma above to strengthen all guards at l_0, l_1 and l_F.

The last guard strengthening we perform makes use of Proposition 6 that says that if the faulty station leaves the active state because of its first and second successor, then the two stations following it do not belong to S_1, but to S_0. In other words, a station from S_1 emitting in the first round following the fault with g cannot be the first nor the second station following the faulty station. Therefore, at location l_1, the guard of transition

$$t_G = i \ \wedge \ C_1 > C_0 \ \wedge \ C_p < N \ \wedge \ r > 0 \ \longrightarrow \ emit!!, t_G^{\oplus\oplus}, C_p^{++}; \ l_1$$

is strengthened into

$t_G = i \ \wedge \ C_1 > C_0 \ \wedge \ C_p < N \ \wedge \ r > 0 \wedge (r \neq 1 \vee \neg g \vee C_p \neq 1 \vee C_p \neq 2)$

without changing the semantics of the program.

We obtain a transformed program $M5$ bisimilar to program $M4$.

4.3.5 Counter abstraction

We are now ready to perform the usual counter abstraction where individual control locations l_{in}, l_0, l_1 and l_F are replaced by counters C_{in}, C_0, C_1 and C_F respectively counting how many processes are at these locations and obtain a single extended finite state machine with a single location, which is then omitted (Delzanno 2000).

Then, we define a new program $M6$ obtained from the program $M5$ by (1) transforming transition $l : g \ \rightarrow \ a; l'$ into $C_l > 0 \ \wedge \ g \ \rightarrow \ a, C_l^{--}, C_{l'}^{++}$, where l, l' are locations or control states and C_l,$C_{l'}$ the associated counters, (2) replacing any condition involving i by *true*, and (3) packing into a single abstract transition all transitions from $M5$ where guards evaluate to true simultaneously.

For example, the two transitions of $M5$ at l_{in}

$$t_G = i \ \wedge \ \neg fault \ \longrightarrow \ emit!!, t_G^{\oplus\oplus}; l_{in}$$

and

$$t_G \neq i \ \wedge \ emit?? \ \wedge \ \neg fault \ \longrightarrow \ ; l_{in}$$

give the abstract transition:

$$C_{in} > 0 \ \wedge \ emit?? \ \wedge \ \neg fault \ \longrightarrow \ emit!!, t_G^{\oplus\oplus}.$$

We obtain program $M6$ shown in Figure 5. The initial value of x guesses the number of processes that move from location l_{in} to l_1.

Such a counter abstraction simulates the previous models and verifies the property that the value of any counter C_l at some state σ' is exactly the number of processes at location l in the corresponding state σ.

Thus, if we show that C_1 or C_0 evaluate to 0 in the program given in Figure 5, we know that set S_0 or S_1 is empty in the concrete model, establishing that there is no clique.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

in N : int, $N > 1$ C_0 : int, init 0 t_G : int, init 1 r : int, init 0
in $fault$: bool. signal C_1 : int, init 1 s_G : int, init 0 d_0 : int, init 0
in g : bool. signal C_p : int, init 1 d_1 : int, init 1 d_F : int, init 0
in x : int, $x \geqslant 1$ C_i : int, init N C_F : int, init 0

loop forever do

$$
\left\{
\begin{array}{l}
[]\ C_i > 0 \ \wedge\ emit?? \ \wedge\ \neg fault \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus} \\
[]\ C_i > 0 \ \wedge\ emit?? \ \wedge\ fault \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, s'_G := t_G, C'_1 := x, C'_0 := N - x, C'_i := 0 \\
[]\ C_0 > 0 \ \wedge\ d_0 < C_0 \ \wedge\ C_0 + C_1 > 2 \times d_1 \ \wedge\ C_p < N \ \wedge\ r = 0 \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_0^{++} \\
[]\ C_0 > 0 \ \wedge\ d_0 = C_0 \ \wedge\ C_0 > C_1 \ \wedge\ C_p = N \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, C'_p := 1, r^{++}, d'_0 := 1, d'_1 := 0, d'_F := 0 \\
[]\ C_0 > 0 \ \wedge\ d_0 < C_0 \ \wedge\ C_0 > C_1 \ \wedge\ C_p < N \ \wedge\ r > 0 \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_0^{++} \\
[]\ C_0 > 0 \ \wedge\ d_0 < C_0 \ \wedge\ C_0 + C_1 \leqslant 2 \times d_1 \ \wedge\ C_p < N \ \wedge\ r = 0 \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_0^{--}, C_F^{++}, d_F^{++} \\
[]\ C_0 > 0 \ \wedge\ d_0 = C_0 \ \wedge\ C_0 \leqslant C_1 \ \wedge\ C_p = N \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C'_p := 1, r^{++}, C_0^{--}, C_F^{++}, d'_0 := 0, d'_F := 1, d'_1 := 0 \\
[]\ C_0 > 0 \ \wedge\ d_0 < C_0 \ \wedge\ C_0 \leqslant C_1 \ \wedge\ C_p < N \ \wedge\ r > 0 \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_0^{--}, C_F^{++}, d_F^{++} \\
[]\ C_1 > 0 \ \wedge\ d_1 < C_1 \ \wedge\ C_0 + C_1 > 2 \times d_0 \ \wedge\ C_p < N \wedge r = 0 \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_1^{++} \\
[]\ C_1 > 0 \ \wedge\ d_1 = C_1 \ \wedge\ C_1 > C_0 \ \wedge\ C_p = N \ \wedge (r \neq 0 \vee \neg g) \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, C'_p := 1, r^{++}, d'_1 := 1, d'_0 := 0, d'_F := 0 \\
[]\ C_1 > 0 \ \wedge\ d_1 = C_1 \ \wedge\ C_1 > C_0 \ \wedge\ C_p = N \ \wedge r = 0 \wedge g \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C'_p := 1, r^{++}, d'_1 := 0, d'_F := 1, d'_0 := 0, C_F^{++} \\
[]\ C_1 > 0 \ \wedge\ d_1 < C_1 \ \wedge\ C_1 > C_0 \ \wedge\ C_p < N \ \wedge\ r > 0 \ \wedge \\
\quad (r \neq 1 \vee \neg g \vee C_p \neq 1 \vee C_p \neq 2) \\
\quad \longrightarrow\ emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_1^{++} \\
[]\ C_1 > 0 \ \wedge\ d_1 < C_1 \ \wedge\ C_0 + C_1 \leqslant 2 \times d_0 \ \wedge\ C_p < N \wedge r = 0 \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_1^{--}, C_F^{++}, d_F^{++} \\
[]\ C_1 > 0 \ \wedge\ d_1 = C_1 \ \wedge\ C_1 \leqslant C_0 \ \wedge\ C_p = N \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C'_p := 1, r^{++}, C_1^{--}, C_F^{++}, d'_F := 1, d'_1 := 0, d'_0 := 0 \\
[]\ C_1 > 0 \ \wedge\ d_1 < C_1 \ \wedge\ C_1 \leqslant C_0 \ \wedge\ C_p < N \ \wedge\ r > 0 \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, C_1^{--}, C_F^{++}, d_F^{++} \\
[]\ C_F > 0 \ \wedge\ d_F < C_F \ \wedge\ C_p < N \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C_p^{++}, d_F^{++} \\
[]\ C_F > 0 \ \wedge\ d_F = C_F \ \wedge\ C_p = N \\
\quad \longrightarrow\ \neg emit!!, t_G^{\oplus\oplus}, C'_p := 1, r^{++}, d'_F := 1, d'_0 := 0, d'_1 := 0
\end{array}
\right.
$$

end loop

Fig. 5. The abstract model $M6$.

4.4 Proving properties

We have used the system given Figure 5, with a slight change concerning rounds, to prove automatically properties of the protocol. We have used an automaton where rounds are represented by states, rather than by the variable r. We make the distinction between the first round after the fault, and later rounds. Condition $r = 0$ is equivalent to the automaton being in state `round1` while $r > 0$ is equivalent to the automaton being in state `later round`. Further, the automaton leaves state `later round` to go to a state `normal` when cliques are stable.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

A first property, called $M1$, that has been proved as true, is that at the end of the first round after the fault:

$$!(C_1 = C_0) \qquad (P1).$$

P1 means that, when the first round after the fault is over, either $|S_1| > |S_0|$ or $|S_0| > |S_1|$, whatever the original partition $\{S_1, S_0\}$ was when the fault occurred.

We have analyzed what leads to $C_1 > C_0$, or $C_0 > C_1$ after 1 round.

First, we have shown that, if $|S_1| > |S_0|$ when the fault occurs, then $C_1 > C_0$ after one round, and vice-versa. Adding the constraint $x > N - x$, we have proved that, at the end of the first round after the fault:

$$(x = C_1) \qquad (P2).$$

Since counters C_1 and C_0 may not increase, this implies $C_1 > C_0$ when $r > 0$. It also implies that all stations from S_1 did send in the first round.

Then we have investigated the case $|S_1| = |S_0|$ when the fault occurs. If set S_1 comes first in the statical order, then $C_1 > C_0$, and vice versa if S_0 comes first. Adding the constraint $x = N - x$ we have proved:

$$AG \ ((d_1 = x \texttt{ and } d_0 < x) \Rightarrow AG\ (C_1 = x)) \qquad (P3),$$

$$AG \ ((d_1 = x \texttt{ and } d_0 < x) \Rightarrow (C_1 + C_0 - 2 * d_1 <= 0)) \qquad (P4).$$

Again, this implies $C_1 > C_0$ after the first round. It also implies that all stations from S_1 did send in the first round.

To prove the main property, we have first shown that, if $r > 0$, when one round is completed, it is not possible to start a new round where both C_1 and C_0 are not 0, i.e., S_1 and S_0 are both not empty.

Indeed the property below is true when $r > 0$:

$$AG\ \neg(\neg(C_1 = 0) \texttt{ and } \neg(C_0 = 0) \texttt{ and } (C_p = N)) \qquad (P6).$$

Finally, we proved that, at the end of the second round after the fault, i.e., when $C_p = N$ and the automaton goes to state `normal`:

$$AG\ (C_1 = 0 \texttt{ or } C_0 = 0) \qquad (P7).$$

$P7$ means that either S_1 or S_0 is empty at the end of the second round. Hence, all active stations have the same membership vectors at the end of the second round and form again a single clique in the graph theoretical sense.

5 Automatic verification: the k faults case

To be able to calculate precisely $CAcc_s$ and $CFail_s$ using global counters only (see Theorem 8) is what makes possible the construction of an abstract model in the 1-fault case. For the k-faults case, the same approach can be taken. Provided that one is able to calculate precisely $CAcc_s$ and $CFail_s$ using global counters, first a behavioural model for a scenario of the k-fault case is constructed, then this behavioural model can be transformed into successive models that simulate each other, replacing local

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

variables with global counters, till the final counter abstraction is obtained and verified automatically, as has been done for the 1-fault case. Thus, what is needed is to establish a generalisation of Theorem 8 for the k-faults case.

5.1 *Calculating* $CAcc_s$ *and* $CFail_s$

Let $1 \leqslant i \leqslant k$. By Proposition 3, after the occurrence of fault i, W, the set of active stations, is partitioned into sets S_w with $w \in \{0,1\}^i$. We find it handy for the following to indicate the length of the string w with the superscript i. We associate two counters Cw^i and dw^i to each set S_{w^i} that is formed after the occurrence of any fault i. The counters Cw^i counts how many stations belong to set S_{w^i} when fault i occurs. The counters dw^i count how many stations from the set S_{w^i} have sent between fault i and fault $i+1$ in case $i < k$, and counts how many stations from the set S_{w^i} have sent so far in the first round following fault k in case $i = k$. Again because of Proposition 3, we assume that, for any $w \in \{0,1\}^{i-1}$, $Cw^{i-1}1 + Cw^{i-1}0 = Cw^i$, $Cw^{i-1}1 \geqslant 1$ and $dw^{i-1}1 \geqslant 1$. Further, for each fault i, we associate a counter $Cp(i)$ that counts how many time slots have elapsed since fault i.

The creation of counters is illustrated taking a particular scenario in Figure 7. After 1 fault, active stations split into two sets, S_0, the stations that have not received the frame correctly and S_1 the stations that have received the frame correctly. Four counters are created: C_{0^1} contains the number of stations from S_0, C_{1^1} contains the number of stations from S_1, d_{0^1} counts the stations from S_0 that are sending frames, d_{1^1} counts the stations from S_1 that are sending frames. $d_{1^1} \geqslant 1$ since the station that was sending when the fault occurred is from S_1. Each time a station from S_1, respectively from S_0 is prevented from sending, then S_1 and C_{1^1}, respectively S_0 and C_{0^1}, are decreased by 1. Suppose that a second fault occurs when some station from S_1 emits. Then S_1 splits into S_{10} and S_{11} and six new counters are created: C_{00^2}, which is, in this scenario, initially equal to the present value of C_{0^1}, C_{10^2}, to count the number of stations in S_{10}, C_{11^2} to count the number of stations in S_{11}, d_{00^2} to count the number of stations from S_{00} (which is, in this scenario, the same as S_0) that are sending frames after the second fault – note that d_{0^1} is not incremented anymore after the occurence of fault 2– d_{10^2} to count the number of stations from S_{10} that are sending frames after the second fault, and d_{11^2} to count the number of stations from S_{11} that are sending frames after the second fault. Again, $d_{11^2} \geqslant 1$ since the station that was sending when the fault occured is from S_{11}. In Figure 6, one assumes that a third fault occurs when a station from set S_{00} is sending and a fourth fault occurs when a station from set S_{100} is sending.

Thus, for k faults and a particular scenario, $\Sigma_{i=1}^{i=k} 2(i+1) + k$ counters so far are created.

These counters are almost enough to know $CAcc_s$ and $CFail_s$ for any station s in the first round following fault k. Indeed, let s be a station ready to send. s belongs to some set S_{w^k}. In the rounds preceding fault k and during the round following fault k, s recognizes as correct frames sent by stations from $S_{w'}$, where w' is a prefix of w^k, and recognizes as incorrect all other frames. This information is recorded with the counters dw^i, $w \in \{0,1\}^i$ and $1 \leqslant i \leqslant k$.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

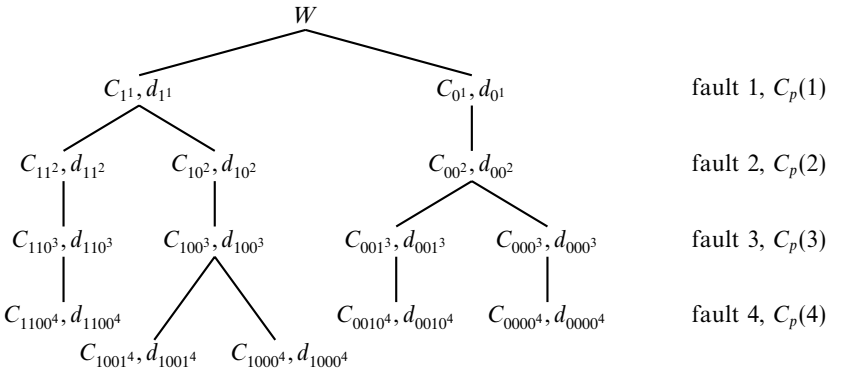


Fig. 6. Illustrating the creation of counters up to four faults.

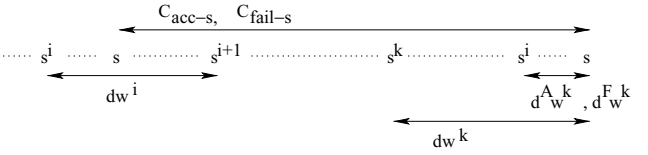


Fig. 7. Evaluating $CAcc_s$ and $CFail_s$ after fault k.

There is one more subtlety. The clique avoidance mechanism needs that $CAcc_s$ and $CFail_s$ count one round only, the round being relative to the position of the sending station s. To do so properly, we distinguish two cases.

The first case is when fault k occurs in the first round following fault 1 and the time slot of s still belongs to that round. One must take into account that s has recognized as correct all stations that have sent before fault 1, which is a straight generalization of Theorem 8. For example, consider the scenario of Figure 6 and a station s from set S_{0010}. Then:

$CAcc_s = |W| - d_{1^1} - d_{11^2} - d_{10^2} - d_{110^3} - d_{100^3} - d_{000^3} - d_{1100^4} - d_{1001^4} - d_{1000^4} - d_{0000^4}$,

$CFail_s = d_{1^1} + d_{11^2} + d_{10^2} + d_{110^3} + d_{100^3} + d_{000^3} + d_{1100^4} + d_{1001^4} + d_{1000^4} + d_{0000^4}$.

The second case is when the time slot of the sending station s lies between station s^i and s^{i+1} and fault k occurs in the first round following fault i, $i > 1$. After fault i, s belongs to some set S_{w^i} and the number of frames accepted as correct by s is given by dw^i. However, to count correctly $CAcc_s, dw^i$ is too much. One has to withdraw all stations accepted by s whose time slots are between s^i and s. This is illustrated in Figure 7. We introduce auxiliary counters $d^A w^k$ and $d^F w^k$. These counters are set to 0 when fault k occurs. Counter $d^A w^k$ counts how many stations from set S_{w^k} have sent so far, as counters dw^k do, and counter $d^F w^k$ counts how many stations from set S_{w^k} were prevented from sending so far by the clique avoidance mechanism and moved to the set of non-working stations. The difference with dw^k is that these counters are reset to 0 each time a counter $Cp(i)$ reaches N after fault k. Thus $dw^i - \Sigma_{w'^k} d^A w'^k - \Sigma_{w'^k} d^F w'^k$, with w^i a prefix of w'^k, gives exactly how many frames between s and s^{i+1} the station s has recognized as correct in the round, and $dw^i - \Sigma_{w'^k} d^A w'^k - \Sigma_{w'^k} d^F w'^k + dw^{i+1} + \cdots + dw^k$ gives exactly how many frames in total s has recognized as correct in the round, i.e., $CAcc_s$. For example, consider

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

again the example illustrated in Figure 6 and a station s from set S_{0010}. Suppose fault 4 occurs in the first round following fault 2 and station s lies between s^2 and s^3, the stations that were emitting when fault 2, respectively fault 3, occurred. Then: $CAcc_s = d_{00^2} + d_{001^3} + d_{0010^4} - d^A_{0010^4} - d^A_{0000^4} - d^F_{0010^4} - d^F_{0000^4}$.

A similar idea works for $CFail_s$. This is formally stated in the proposition below.

Proposition 9

We indicate by w_s that the string w refers to an entity where s belongs.

1. Let $s \in S_{w^k_s}$ a station ready to send in the round following fault k.
   (a) If $Cp(1) \leqslant N$ at the time slot of s, then:
   $CAcc_s = \mid W \mid - \Sigma_{w'} dw'$, and $CFail_s = \Sigma_{w'} dw'$,
   where w' must not be a prefix of w^k_s.
   (b) Let $1 < i < k$ such that $Cp(i) \geqslant N$ and $Cp(i+1) < N$ at the time slot of s. Then:
   $CAcc_s = (\Sigma_{j=i}^{j=k} dw^j_s) - \Sigma_{w'^k} d^A w'^k - \Sigma_{w'^k} d^F w'^k$, where w^j_s is a prefix of w^k_s and w^i_s is a prefix of all w'^k, and
   $CFail_s = (\Sigma_{j=i}^{j=k} \Sigma_{w^j \neq w^j_s} dw^j) - \Sigma_{w^k \neq w^k_s} d^A w^k - \Sigma_{w^k \neq w^k_s} d^F w^k$,
   where w^k must be a suffix of some $w^i \neq w^i_s$.
2. Let $s \in S^k_w$ a station ready to send in the second round following fault k. Then:
   $CAcc_s = dw^k - d^F w^k$, and
   $CFail_s = \Sigma_{w'^k} dw'^k - \Sigma_{w'^k} d^F w'^k$ with $w'^k \neq w^k$.

Proof

For 1, the proof follows what has been exposed informally above. For 2, one assumes counters dw^k are kept as there are at the end of the first round, and that $d^F w^k$ are reset to 0 and incremented during the second round each time some station from set S_{w^k} is prevented from sending. The result follows, since dw^k contains exactly the number of stations from set S_{w^k} that have sent in the first round following fault k. □

5.2 Complexity issues

It follows that, for one scenario of the k faults case, the total number of counters needed is $\Sigma_{i=1}^{i=k} 2(i+1) \ + k + 2(k+1)$. There is $\Pi_{i=1}^{i=k} i$ possible scenarios to check.

Using all these counters, an extended automaton similar to the one given in Figure 5 can be designed and, in theory, automatically verified. Properties analogous to $P6$ and $P7$ have to be checked to prove that after the second round following fault k, there is only 1 clique. However, in practice, tools that are presently available do not make it possible to handle such a number of counters already for two faults[1]. Though, a scenario for two faults has been successfully verified in Bardin *et al.* (2004) using their tool FAST after performing further ad hoc abstractions to reduce the number of counters.

[1] In the case of two faults, we got memory problems both with ALV and LASH.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

6 Conclusion

We have proposed an approach for verifying automatically a complex algorithm which is industrially relevant. The complexity of this algorithm is due to its very subtle dynamic which is hard to model. We have shown that this dynamic can be captured by means of unbounded (parametric) counter automata. Even if the verification problem for these infinite-state models is undecidable in general, there exists many symbolic reachability analysis techniques and tools which allow to handle such models.

Our approach allows to build a model (counter automaton) for the algorithm with an arbitrary number n of stations, but for a given number k of faults. We have experimented our approach by verifying in a fully automatic way the model in the case of one fault, using the ALV tool and the LASH tool.

Related work

Bauer and Paulitsch (2000) provides a manual proof of the algorithm in the 1 fault case. Theorem 4 generalizes this result to the case of any number of faults. As far as we know, all the existing works on automated proofs or verifications of the membership algorithm of TTP concern the case of one fault, and only symmetric fault occurrences are assumed. In our work, we consider the more general framework where several faults can occur, and moreover, these faults can be asymmetric. In Pfeifer (2000), a mechanised proof using PVS is provided. Katz *et al.* (1997), Baukus *et al.* (2000) and Creese and Roscoe (1999) adopt an approach based on combining abstraction and finite-state model-checking. Katz *et al.* (1997) has checked the algorithm for six stations. Baukus *et al.* (2000) and Creese and Roscoe (1999) consider the parametric verification of n stations; Creese and Roscoe (1999) provide an abstraction proved manually, whereas Baukus *et al.* (2000) uses an automatic abstraction generation technique, both abstractions leading to a finite-state abstraction of the parameterized network. The abstractions used in those works seem to be non-extensible to the case of asymmetric faults and to the k faults case. To tackle this more general framework, we provide an abstraction which yields a counter automaton and reduce the verification of the algorithm to the symbolic reachability analysis of the obtained infinite-state abstract model. Moreover, our abstraction is exact in the sense that it models faithfully the emission of frames by stations.

Future work

Our future work is to automatize, for instance using a theorem prover, the abstraction proof which allows to build the counter automaton modeling the algorithm. More generally, an important issue is to design automatic abstraction techniques allowing to produce infinite-state models given by extended automata. Finally, a challenging problem is to design an algorithmic technique allowing to verify automatically the algorithm by taking into account simultaneously both of its parameters, i.e., for any number of stations *and for any number of faults*.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

Acknowledgments

We thank anonymous referees for their helpful comments.

References

Abdulla, P., Annichini, A., Bensalem, S., Bouajjani, A. and P. Habermehl, Y. L. 1999. Verification of Infinite-State Systems by Combining Abstraction and Reachability Analysis. *Computer Aided verification CAV'99*. Lecture Notes in Computer Science, vol. 1633. Springer Verlag, pp. 146–159.

Abdulla, P. A. and Jonsson, B. 2001a. Channel representations in protocol verification. *CONCUR'01*. Lecture Notes in Computer Science, vol. 2154. Springer Verlag, pp. 1–15.

Abdulla, P. A. and Jonsson, B. 2001b. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science 256*, 145–167.

Annichini, A., Asarin, E. and Bouajjani, A. 2000. Symbolic techniques for parametric reasoning about counter and clock systems. *Intern. Conf. on Computer Aided Verification (CAV'00)*. Lecture Notes in Computer Science, vol. 1855. Springer Verlag, pp. 419–434.

Bardin, S., Finkel, A. and Leroux, J. 2004. Faster acceleration of counter automata in practice. *Intern. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*. Lecture Notes in Computer Science, vol. 2988. Springer Verlag, pp. 576–590.

Bauer, G. and Paulitsch, M. 2000. An investigation of membership and clique avoidance in ttp/c. *Proceedings 19th IEEE Symposium on Reliable Distributed Systems (SRDS'00)*. IEEE Computer Society. 118–124.

Baukus, K., Lakhnech, Y., and Stahl, K. 2000. Verifying universal properties of parameterized networks. *Proceedings of the 5th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems, FTRTFT 2000, Pune, India*, pp. 291–303.

Benveniste, A. and Berry, G. 1991. The synchronous approach to reactive and real-time systems. *Proc. IEEE 79*(9), 1270–1282.

Boigelot, B. and Wolper, P. 1994. Symbolic Verification with Periodic Sets. *Computer Aided verification CAV'94*. Lecture Notes in Computer Science, vol. 818. Springer Verlag, pp. 55–67.

Bouajjani, A., Esparza, J., and Maler, O. 1997. Reachability Analysis of Pushdown Automata: Application to Model Checking. *International Conference on Concurrency Theory CONCUR'97*. Lecture Notes in Computer Science, vol. 1243. Springer Verlag, pp. 135–150.

Bouajjani, A. and Habermehl, P. 1997. Symbolic Reachability Analysis of FIFO-Channel Systems with Nonregular Sets of Configurations. *ICALP'97*. Lecture Notes in Computer Science, vol. 1256. Springer Verlag. (Full version in *Theoretical Computer Science 221*(1/2), 221–250, 1999.)

Bouajjani, A., Jonsson, B., Nilsson, M. and Touili, T. 2000. Regular model checking. *Computer Aided verification CAV'00*. Lecture Notes in Computer Science, vol. 1855. Springer Verlag, pp. 403–418.

Bouajjani, A. and Merceron, A. 2002. Parametric verification of a group membership algorithm. *7th Intern. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02)*. Lecture Notes in Computer Science, vol. 2469. Springer-Verlag, Oldenburg, Germany, pp. 311–330.

Bultan, T., Gerber, R., and League, C. 1998. Verifying systems with integer constraints and Boolean predicates: A composite approach. *Proc. of the Intern. Symp. on Software Testing and Analysis*. ACM Press, pp. 113–123.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press

Bultan, T. and Yavuz-Kahveci, T. 2001. Action language verifier. *Proceedings of the 16th IEEE International Conference on Automated Software Engineering (ASE 2001), Coronado Island, California*. IEEE Computer Society, pp. 382–386.

Cousot, P. and Halbwachs, N. 1978. Automatic discovery of linear restraints among variables of a program. *POPL'78*, ACM, pp. 84–96.

Creese, S. and Roscoe, A. W. 1999. Ttp: a case study in combining induction and data independence. *Oxford University Programming Research Group*. Technical Report TR-1-99.

Delzanno, G. 2000. Automatic Verification of Parameterized Cache Coherence Protocols. *Computer Aided verification CAV'00*. Lecture Notes in Computer Science, vol. 1855. Springer Verlag, pp. 53–68.

Graf, S. and Saidi, H. 1997. Construction of abstract state graphs with pvs. *Computer Aided verification CAV'97*. Lecture Notes in Computer Science, vol. 1254. Springer Verlag, pp. 72–83.

Katz, S., Lincoln, P. and Rushby, J. 1997. Low-overhead Time-Triggered Group Membership. *International Workshop on Distributed Algorithms WDAG'97*. Lecture Notes in Computer Science, vol. 1320. Springer Verlag, pp. 155–169.

Kesten, Y., Maler, O., Marcus, M., Pnueli, A. and Shahar, E. 1997. Symbolic Model Checking with Rich Assertional Languages. *Computer Aided verification CAV'97*. Lecture Notes in Computer Science, vol. 1254. Springer Verlag. (Full version in *Theoretical Computer Science 256*, 93–112.)

Kopetz, H. 1999. Ttp/c protocol – specification of the ttp/c protocol. `http://www.tttech.com (1999)`.

Kopetz, H. and Grünsteidl, G. 1999. A time triggered protocol for fault-tolerant Real-Time Systems. *IEEE Computer*, 14–23.

LASH 2002. The Liège Automata-based Symbolic Handler (LASH). Available at `http://www.montefiore.ulg.ac.be/~boigelot/research/lash/`.

Manna, Z. and Pnueli, A. 1995. *Temporal Verification of Reactive Systems–Safety*. Springer-Verlag.

Pfeifer, H. 2000. Formal verification of the TTP Group Membership Algorithm. *International Conference on Formal Description Techniques for Distributed Systems and Communication protocols and Protocol Specification, Testing and Verification, FORTE/PSTV 2000, Pisa, Italy*, pp. 3–18.

Pnueli, A., Xu, J. and Zuck, L. 2002. Liveness with (0,1,∞)-Counter Abstraction. *Computer Aided verification CAV'02*. Lecture Notes in Computer Science, vol. 2404. Springer Verlag, pp. 107–122.

Saidi, H. and Shankar, N. 1999. Abstract and Model Check while you Prove. *Computer Aided verification CAV'99*. Lecture Notes in Computer Science, vol. 1633. Springer Verlag, pp. 443–454.

Steiner, W., Rushby, J., Sorea, M. and Pfeifer, H. 2004. Model checking a fault-tolerant startup algorithm: From design exploration to exhaustive fault simulation. *Proc. of the International Conference on Dependable Systems and Networks DSN 2004*, pp. 189–198.

Wolper, P. and Boigelot, B. 1998. Verifying systems with infinite but regular state spaces. *Computer Aided verification CAV'98*. Lecture Notes in Computer Science, vol. 1427. Springer Verlag, pp. 88–97.

https://doi.org/10.1017/S1471068406002663 Published online by Cambridge University Press