



ELSEVIER

Science of Computer Programming 29 (1997) 171–197

**Science of
Computer
Programming**

Specification and verification of various distributed leader election algorithms for unidirectional ring networks¹

Hubert Garavel^{a,*}, Laurent Mounier^b

^a INRIA Rhône-alpes, 655 avenue de l'Europe, 38330 Montbonnot St Martin, France

^b VERIMAG, Zone industrielle de Mayencin, 2 rue de Vignate, 38610 Gières, France

Abstract

This paper deals with the formal specification and verification of distributed leader election algorithms for a set of machines connected by a unidirectional ring network.

Starting from an algorithm proposed by Le Lann in 1977, and its variant proposed by Chang and Roberts in 1979, we study the robustness of these algorithms in the presence of unreliable communication medium and unreliable machines. We propose various improvements of these algorithms in order to obtain a fully fault-tolerant protocol.

These algorithms are formally described using the ISO specification language LOTOS and verified (for a fixed number of machines) using the CADP (CÆSAR/ALDÉBARAN) toolbox. Using model-checking and bisimulation techniques, the verification of these non-trivial algorithms can be carried out automatically, in a few seconds. © 1997 Elsevier Science B.V.

Keywords: Distributed algorithms; Formal methods; Leader election; LOTOS; Token ring; Validation; Verification

1. Introduction

Since the early attempts of automatic protocol validation carried out by West [48], formal verification in general, and the so-called model checking approach in particular, is sometimes controversial:

- From a theoretical point of view, this problem can be considered as solved for finite state systems since verification algorithms are now well known;
- From a naive point of view, formal verification is useless, since no realistic application can be fully verified, and this problem is untractable anyway because of the size of realistic systems.

* Corresponding author. Tel.: +(33) 4 76 61 52 24; fax: +(33) 4 76 61 52 52; e-mail: hubert.garavel@imag.fr, laurent.mounier@imag.fr.

¹This work has been supported in part by the European Commission, under project ISC-CAN-65 “EUCALYPTUS-2: A European/Canadian LOTOS Protocol Tool Set”.

Fortunately, reality is more shaded: practical experiments already showed that these methods are now applicable in an industrial context, and that they can provide an important gain in the design of a large class of applications, such as distributed systems or communication protocols.

This paper illustrates the application of state-of-the-art verification methods to a non-trivial case-study, hardly tractable simply by human reasoning. Starting from the original description of two famous algorithms, the one proposed by Le Lann [36] and its variant proposed by Chang and Roberts [7], our objective was to design a fully fault-tolerant leader election algorithm, to specify it using the LOTOS language, and to formally verify it using the CADP toolbox.

This paper is organized as follows. Section 2 presents the LOTOS language and Section 3 gives an overview of the CADP toolbox. We then study several versions of the leader election algorithm under increasingly severe failure assumptions: reliable links and reliable machines (Section 4), unreliable links and reliable machines (Section 5) and, finally, unreliable links and unreliable machines (Section 6).

2. The formal description technique LOTOS

Many formalisms have been proposed for describing parallel systems, among which, the standardized Formal Description Technique LOTOS² has received a considerable attention from the research community.

LOTOS is a formal language intended for the specification of communication protocols and distributed systems. It was developed during the years 1981–88 in the framework of the SEDOS³ project and standardized by Iso⁴ in 1988 [22]. Several tutorials for LOTOS are available, e.g. [2, 46].

The design of LOTOS was motivated by the need for a language with a high abstraction level and a strong mathematical basis, which could be used for the description and analysis of complex systems. As a design choice, LOTOS consists of two distinct and “orthogonal” sub-languages:

- **the data part** of LOTOS is dedicated to the description of data structures. It is based on the well-known theory of algebraic abstract data types, more specifically on the ACTONE specification language [8].
- **the control part** of LOTOS is based on the process algebra approach for concurrency, and appears to combine the best features of CCS [37, 38] and CSP [20].

LOTOS has been applied to describe complex systems formally, among which: the service and protocols for the OSI transport and session layers [24, 23, 28, 29], the CCR⁵

² Language Of Temporal Ordering Specification.

³ Software Environment for the Design of Open Distributed Systems, ESPRIT project 410.

⁴ International Organization for Standardization.

⁵ Commitment, Concurrency, and Recovery.

service and protocol [33, 32], OSI TP⁶ [27, Annex H], MAA⁷ [26, 40], FTAM⁸ basic file protocol [21, 34], etc. It has been mostly used to describe software systems, although there are several attempts to use it for asynchronous hardware description.

A number of tools have been developed for LOTOS, which cover most user needs in the areas of simulation, compilation, test generation and formal verification. We have used the CADP toolbox [12, 11], which provides state-of-the-art verification features.

3. Description of the verification tools used

The CADP⁹ toolbox is dedicated to the design and verification of communication protocols and distributed systems. Initiated in 1986, several motivations have contributed to its development since this date:

- This toolbox aims to offer an integrated set of functionalities ranging from interactive simulation to exhaustive model-based verification methods. In particular, both logical and behavioural specifications can be verified.
- One of the major objectives of the toolbox is to deal with large case studies. Therefore, as well as classical enumerative verification methods, it also includes more sophisticated approaches, such as symbolic and on-the-fly verification, and compositional model generation.
- Finally, this toolbox can be viewed as an open software platform: in addition to LOTOS, it also supports low-level formalisms such as finite state machines and networks of communicating automata.

In the sequel we only present the components used throughout this case-study:

- CÆSAR [16] and CÆSAR.ADT [15, 17] are compilers able to translate a LOTOS program into a finite state graph (or Labelled Transition System) describing its exhaustive behaviour. This graph can be either *explicitly* represented as set of states and transitions, or *implicitly*, namely as a C library providing a set of functions allowing to execute the program behaviour in a controlled way.
- ALDÉBARAN [10, 13] is a verification tool able either to compare or to minimize graphs with respect to (bi)simulation relations [41, 37]. Initially designed to deal with explicit graphs produced by CÆSAR, it has been extended to also handle networks of communicating automata (for on-the-fly and symbolic verification).
- TERMINATOR, EXHIBITOR, XSIMULATOR, and EVALUATOR, respectively, provide partial deadlock detection, incorrect execution sequences exhibition, interactive simulation, and evaluation of temporal logic formulas. These tools operate on-the-fly and do not require to construct an explicit graph first; they can therefore be applied to large programs.

⁶ Distributed Transaction Processing.

⁷ Message Authentication Algorithm.

⁸ File Transfer, Access and Management.

⁹ CÆSAR/ALDÉBARAN Development Package.

4. Token-passing with reliable links and reliable stations

4.1. Problem statement and expected properties

We consider the well-known problem of a system with n machines (hereafter called *stations* and noted S_1, \dots, S_n , respectively) sharing a common resource R . Each station S_i is given a unique identifier, its *address* A_i . For each station, we define two events “OPEN ! A_i ” and “CLOSE ! A_i ”, which respectively take place when S_i starts to access R (resource acquisition) and when S_i stops to access R (resource liberation). We assume here that all stations are reliable and we will not consider the possibility of station crashing until Section 6.1.

We want to design the behaviour of each station in a such a way that the global system satisfies several “good” properties. According to OSI¹⁰ terminology, we call *protocol* the behaviour of the stations and *service* the conjunction of the “good” properties. For these properties, we decide to abstract all details of the system except the “OPEN ! A_i ” and “CLOSE ! A_i ” events, as we are only interested in observing accesses to the shared resource. We want to ensure two “good” properties:

- The first one is *mutual exclusion* between stations accessing the resource: each interval between “OPEN ! A_i ” and “CLOSE ! A_i ” should be a critical section. Mutual exclusion is a safety property¹¹ [35, 1]. It does not imply deadlock freedom (a blocked process satisfies all possible safety properties, including mutual exclusion).
- The second one is *equal opportunity*: at any time, there exists an execution sequence allowing each station to access the resource before any other station. Equal opportunity is a liveness property¹² [35, 1]. In particular, under the usual fairness assumption given in [44], equal opportunity implies that no station can be indefinitely denied access to the resource. It also implies deadlock freedom, as each station always remains active to access the resource. However, it does not imply livelock freedom, as the fairness assumption abstracts away infinite loops of invisible actions (due to indefinite losses of messages, for instance). Notice that a protocol that would serialize the resource accesses granted to the stations (by establishing priorities between stations or by implementing a round-robin policy, for instance) would not satisfy the equal opportunity property.

The approach we used to verify that the protocol satisfies the service properties consists in expressing the service as a (set of) graph(s), generating a graph corresponding to the behaviour of the protocol, and comparing the protocol graph against the service graph(s) modulo appropriate *equivalence or preorder* relations. In this approach, it is first necessary to determine which comparison relation(s) to use and to express service graph(s) according to this(ese) relation(s).

¹⁰ Iso standards for Open System Interconnection.

¹¹ Safety properties assert that “something bad never happens”. They can be characterized by the class of prefix-closed properties: P is a safety property iff for each finite execution sequence Σ satisfying P , all the prefixes of Σ also satisfy P .

¹² Liveness properties assert that “something good eventually happens”.

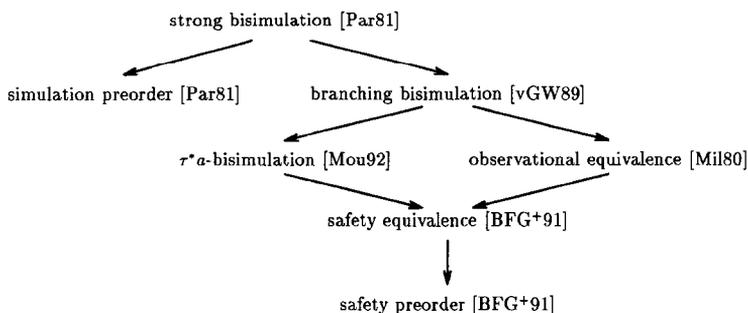


Fig. 1. Lattice of the relations currently implemented in ALDÉBARAN.

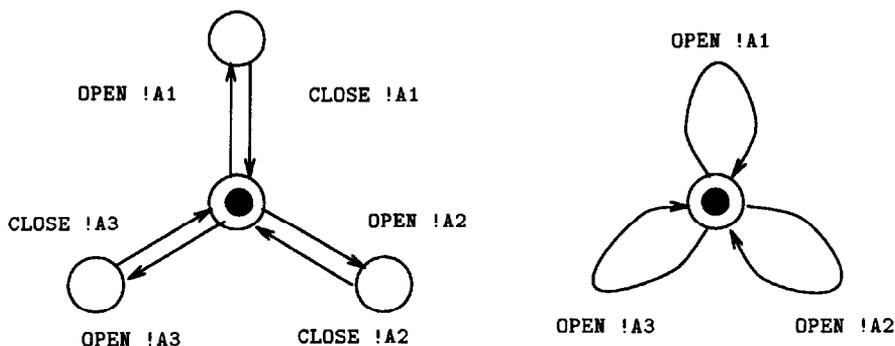


Fig. 2. Graphs expressing mutual exclusion and equal opportunity.

Many relations, especially *bisimulations*, have been defined in the literature. A tentative classification of bisimulation can be found in [39]; Fig. 1 shows the lattice of the bisimulation relations which are currently implemented in ALDÉBARAN tool: the edges of the diagram go from the strongest to weakest relations. These relations present subtle differences: consequently, the determination of the appropriate(s) relation(s) for a given problem often requires some expertise in bisimulations.

The service we consider here consists in both mutual exclusion and equal opportunity:

- As mutual exclusion is a safety property, a natural candidate to verify it is safety equivalence [3] which preserves all safety properties. The graph on the left of Fig. 2 is the minimal graph (in number of states) expressing mutual exclusion for three stations with respect to safety equivalence.¹³
- Equal opportunity is a stronger property, which is preserved neither by safety equivalence nor by τ^*a -bisimulation (for instance, deadlocks are not preserved in general

¹³ More precisely, any graph smaller than this graph with respect to the safety preorder satisfies mutual exclusion.

by these two relations). Both remaining candidates, observational equivalence and branching bisimulation¹⁴ preserve deadlocks. A way to express this property consists in hiding, in the protocol graph, all events but “OPEN ! A_i ” and to compare the resulting graph to the graph given on the right of Fig. 2 modulo either branching or observational equivalence.

Although mutual exclusion and equal opportunity can be verified separately, it would be nice to check them all at once, using a single equivalence relation and a single service graph. Noticing that both graphs on Fig. 2 are branching equivalent when “CLOSE ! A_i ” are hidden, it is clear that any protocol graph branching equivalent to the graph on the left of Fig. 2 satisfies both mutual exclusion and equal opportunity.¹⁵

The graph on the left of Fig. 2 (called the *service graph* in the sequel) can also be described by a LOTOS process, which behaves as follows: it chooses an address A_i non-deterministically, then performs an “OPEN ! A_i ” event followed by a “CLOSE ! A_i ” event; finally the process goes back into its initial state (in LOTOS this is expressed using a recursive process call). The “noexit” keyword indicates that process “SERVICE” never terminates (since it loops forever).

```

1 process SERVICE [OPEN, CLOSE] : noexit :=
2   choice Ai:ADDR []
3   (
4     OPEN !Ai;
5     CLOSE !Ai;
6     SERVICE [OPEN, CLOSE]
7   )
8 endproc

```

4.2. Token-based protocols

The definition of the service given above is very abstract and only provides a requirement for a proper functioning of the protocol (it specifies “what” to do, but not “how”). It is still necessary to define the protocol allowing stations to access the shared resource according to service constraints. This problem occurs for Local Area Networks, in which several stations compete for the “right to speak” on a common medium. Various protocols belonging to the *Medium Access Control* part of the physical layer of the OSI model (see, e.g., [45]) have been designed for this purpose. Two of them assume a circular organization of stations on a ring, either a physical ring (as in the *token ring* protocol [31]) or a logical ring, build upon a tree-like physical network (as in the *token bus* protocol [25]). Some other protocols, such as FDDI¹⁶ [30], rely on a more complex, double-ring topology.

¹⁴ Strong equivalence is not useful for verifying graphs which contain invisible (i.e., τ) actions.

¹⁵ This is also true for observation equivalence, but we adopt here branching equivalence for two reasons: when the service graph contains no τ -transitions, both relations coincide [39]; moreover, the algorithms for branching bisimulation [19, 13] are more efficient.

¹⁶ Fibre Distributed Data Interface.

Roughly speaking, in token-based protocols, a particular message (called the *token*) circulates permanently on the ring. At a given instant, only the station that owns the token is allowed to access the shared resource.

We will consider a network of n stations circularly placed on a (simple) ring. Although it could seem natural, at first sight, to interconnect neighbour stations directly using LOTOS rendez-vous, we need to insert one auxiliary process between each pair of neighbour stations. There are n such auxiliary processes in the ring; they are called *links* and noted L_i . As discussed in Sections 4.3 and 5.1, they are needed to model accurately the properties of the communication channels connecting the stations.

Due to the finite state verification methods we use, the verification is necessarily partial, in the sense that it can only be performed for a fixed number of stations and links. In the sequel, we will consider the case of a ring with three stations and three links. Choosing the value $n = 3$ is convenient for us to present all the different verification methods used in the CADP toolbox, starting from the simplest ones (brute-force graph generation, used before Section 5.4) up to the more sophisticated ones (compositional, on-the-fly, and symbolic techniques, used since Section 5.4). The issue of generalizing our results for $n > 3$, or even for all values of n , will be discussed in Section 7.

We also assume that all stations and all links behave (almost) identically. Their respective behaviours will be defined as particular instances of two generic LOTOS processes, “STATION” and “LINK”.

4.3. Reliable links

As the LOTOS rendez-vous mechanism is fully symmetric, the ring architecture defined above could very well support bidirectional communication, allowing messages to circulate in both ways on the ring. To model a uni-directional communication channel, symmetry must be broken. This is a reason for introducing link processes in our formal description.

We assume that all links preserve message ordering (*FIFO property*), i.e., messages are received in the same order as they are emitted. We model each link as a one-slot buffer, with a cyclical behaviour: it receives a message on its input gate, then transmits this message on its output gate, and returns to its initial state.

We first assume that links carry a simple type of message, noted “!TOKEN”, where “TOKEN” is a value of some enumerated type. We obtain the following description in LOTOS for the reliable link:

```
1 process LINK [INPUT, OUTPUT] : noexit :=
2   INPUT !TOKEN;
3   OUTPUT !TOKEN;
4   LINK [INPUT, OUTPUT]
5 endproc
```

4.4. Basic stations

We now consider a station (called the *basic station*) that implements a very simple token-passing protocol. The behaviour of this station can be explained as follows. From its initial state, the station can immediately go to a *privileged state*, noted π , if its “INIT” variable is equal to *true*; if not, the station must wait until the token is received before going to state π . In state π , the station can access the shared resource and then pass the token to the next station; it can also pass the token directly, without accessing the resource. In both cases, when passing the token, the station loses its privilege, sets variable “INIT” to *false*, and returns to its initial state. Resource accesses by station S_i are modelled by two consecutive events “OPEN ! A_i ” and “CLOSE ! A_i ”: as we are only interested in mutual exclusion and equal opportunity, we do not otherwise model the actions performed by the station when accessing the resource.

The basic station can be described in LOTOS by two mutually recursive processes shown below. We introduce a “PRIVILEGE” process that expresses the behaviour of the station in state π : this process is useful for factorization purpose and will also be present in more sophisticated versions of the protocol. The LOTOS operator “ \rightarrow ” is used to test the value of variable “INIT”. As LOTOS does not allow direct variable assignment, the value of variable “INIT” has to be modified indirectly, using parameter passing.

```

1 process STATION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, INIT:BOOL) : noexit :=
2   [INIT = true] ->
3     PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
4   []
5   [INIT= false] ->
6     PRED !TOKEN;
7     PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
8 endproc

9 process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR) : noexit :=
10  SUCC !TOKEN;
11  STATION [OPEN, CLOSE, PRED, SUCC] (Ai, false)
12  []
13  OPEN !Ai;
14  CLOSE !Ai;
15  SUCC !TOKEN;
16  STATION [OPEN, CLOSE, PRED, SUCC] (Ai, false)
17 endproc

```

Using the CADP toolbox, the ring consisting of three basic stations connected by three reliable links was proven to be branching equivalent to the service graph. This was done by generating the graph of the ring using CÆSAR and CÆSAR.ADT, and by comparing this graph to the one shown on the left of Fig. 2 using ALDÉBARAN. Quantitative details about this verification can be found in Appendix A under reference #1.

The CADP toolbox was also used to study two “pathological” situations in which all stations are initialized with $\text{INIT} = \textit{false}$, or several stations are initialized with $\text{INIT} = \textit{true}$. The results obtained confirm the intuition: in the former case, a deadlock is observed; in the latter case, mutual exclusion property is violated.

5. Token-passing with unreliable links and reliable stations

We now consider a first kind of network failure: the possibility of having unreliable links, i.e., links that can randomly loose messages. This assumption is realistic if we try to apply token-passing mechanism in the context of metropolitan area networks with possibly unreliable communications. For instance, the token ring standard [31] allows any kind of point-to-point physical medium to connect the stations.

5.1. Unreliable links

Informally, an unreliable link should behave as follows: it accepts a message on its “INPUT” gate and decides *internally* to deliver this message on its “OUTPUT” gate, or to loose this message silently. There are several possible modelings of unreliable links, which differ by their branching structure, especially with respect to non-determinism and invisible actions τ . We chose the following one, whose behaviour includes the behaviour of the reliable link when synchronized with an arbitrary environment:

```

1 process LINK [INPUT, OUTPUT] : noexit :=
2   INPUT !TOKEN;
3   OUTPUT !TOKEN;
4   LINK [INPUT, OUTPUT]
5   []
6   INPUT !TOKEN;
7   LINK [INPUT, OUTPUT]
8 endproc
```

5.2. Le Lann’s and Chang–Roberts’ leader election algorithms

We now consider the problem of designing stations that can recover from token losses, i.e., finding a fault-tolerant algorithm that works reliably even in presence of unreliable links. The basic idea for recovery is quite simple: “every time a token is lost, a new token must be generated”, but it raises several issues:

- The algorithm should be able to detect token losses, which is not obvious because links can loose tokens silently; neither the station that sent the token nor the station that expects to receive it are informed of a loss. The standard solution for this problem is based on *timeout*: a station that did not receive the token for a while might assume that the token was lost; this is usually triggered by the expiration of a timer. However, occurrence of timeout does not guarantee that the token was actually lost, since we make no assumption on the relative speeds of stations and links, nor on the time each station can spend in the critical section. We only consider *time-independent* algorithms, i.e., algorithms that work correctly for any value of the timers, rather than algorithms whose correctness rely on a particular tuning of timers.
- There should be at least one token in the ring (otherwise, deadlock ensues) and at most one token (otherwise mutual exclusion is violated). Therefore, the algorithm

should ensure that a new token – and a single one – is generated each time a loss is detected.

The simplest solution to this problem assumes the existence of a distinguished station (called the *monitor*) that is responsible for generating a single new token when the current one gets lost. This solution forms the basis of the token ring protocol [31]. However, if we assume that all stations, including the monitor, can be unreliable (a problem that will be considered in Section 6), this simple solution is not sufficient. For this reason, the token ring protocol uses a more sophisticated algorithm (often referred to as *leader election*) to recover from failures of the monitor.

In general, leader election algorithms aim at identifying a single object in a group of objects. We consider here a particular case, in which these algorithms are applied to a group of stations connected by a circular ring. Because of the mutual exclusion constraint, the problem is more complex than a simple leader election: several elections can take place sequentially (or even simultaneously), and a leader may already exist at the time an election is started.

The first leader election algorithm for a unidirectional ring network was proposed in 1977 by Gérard Le Lann [36]. Its principles can be summarized as follows:

- When the token is lost, all stations have to elect unanimously one station that will generate a new token.
- A total order relation “ $<$ ” is defined over the station addresses A_i . The election rule is such that the functioning station with the smallest address will be elected.
- When a station believes that the token was lost (this is triggered by the expiration of a timer), it starts an election (or participates to an already started election) by sending a candidature message.¹⁷ Candidatures circulate around the ring and have the form “!CLAIM ! A_i ”, where “CLAIM” is some value of an enumerated type and A_i is the address of the station that issued the candidature. A station becomes eligible after sending its candidature.
- When a station receives a candidature stamped with a smaller address than its own address, it transmits this candidature and ceases to be eligible if it was.
- When a station receives a candidature stamped with a greater address than its own address, it transmits this candidature and remains eligible if it was.
- When a station receives a candidature stamped with its own address, if it is eligible, it becomes privileged and generates a new token; otherwise, it simply discards the candidature.
- Initially, there is no token in the ring and no station is privileged: the stations will spontaneously start an election to determine which station will generate the first token.

In [36], the behaviour of each station is defined by a state machine with four states (noted α , β , γ , and α^*) and a list of transitions triggered by message (token or claim) receipts.

¹⁷ Our terminology slightly differs from Le Lann’s one since we use “token” instead of “control token”, and “claim” or “candidature” instead of “(candidate) token”.

However, when trying to formalize in LOTOS this behaviour, several ambiguities appeared, for which we had to bring solutions:

- Only message receipts are specified, nothing is said about messages that stations can (or must) send in a given state. We therefore added two message emissions: the token is sent when the privileged station leaves the critical section, and claims emitted by other stations are transmitted to the next station.
- Le Lann's algorithm does not model the accesses to the shared resource. Moreover, the state α^* is really unclear and raises many questions (What is the meaning of "immediate switching"? What happens to messages received in this state, especially timer awakening?) We replaced α^* with a privileged state π , which is reached when the station receives the token from its neighbour or becomes elected. In state π the station behaves as the process "PRIVILEGE" described in Section 4.4.
- As the value of timers is not important, we modelled timer awakening followed by claim emission by a simple claim emission (which is equivalent from the environment point of view).

The LOTOS description corresponding to Le Lann's algorithm is the following:

```

1 process STATION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, INIT:BOOL) : noexit :=
2   ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)
3 endproc

4 process ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, S:STATE):noexit :=
5   SUCC !CLAIM !Ai; (* timeout *)
6   ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, BETA)
7   []
8   PRED !TOKEN;
9   PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
10  []
11  PRED !CLAIM ?Aj:ADDR;
12  (
13    [Ai < Aj] ->
14      SUCC !CLAIM !Aj;
15      ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S)
16    []
17    [Ai > Aj] ->
18      SUCC !CLAIM !Aj;
19      (
20        [S == BETA] ->
21          ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, GAMMA)
22        []
23        [S <> BETA] ->
24          ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S)
25      )
26    []
27    [Ai == Aj] ->
28      (
29        [S == BETA] ->
30          PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
31        []
32        [S <> BETA] ->
33          ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)

```

```

34         )
35     )
36 endproc

37 process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR): noexit :=
38     OPEN !Ai;
39     CLOSE !Ai;
40     SUCC !TOKEN;
41     ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)
42 []
43     SUCC !TOKEN;
44     ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)
45 endproc

```

In 1979, Ernest Chang and Rosemary Roberts proposed an improvement [7] to Le Lann's algorithm, based on the following observation: when a station S_i receives a “weak” candidature (i.e., a candidature stamped with an address A_j such that $A_i < A_j$), station S_i knows for sure that station S_j may not win the election.¹⁸ The “weak” candidature is therefore useless and can be “filtered” by station S_i , meaning that S_i does not transmit this candidature to the next station. As each station only transmits “strong” candidatures to the next station, it is clear that the number of messages exchanged during an election is reduced.¹⁹ If there are n stations, Le Lann's algorithm requires $O(n^2)$ messages in the average case, whereas Chang–Roberts' algorithm requires $O(n \log n)$ messages in the average case.²⁰ Practically, Chang–Roberts' algorithm can be directly derived from Le Lann's algorithm by removing line 14 of the above LOTOS text.

We first tried to verify automatically a ring consisting of three Le Lann's stations connected with three reliable links.²¹ We generated the corresponding graph using CÉSAR and compared it using ALDÉBARAN against the service graph (see Appendix A under reference #3). ALDÉBARAN revealed that both graphs were not branching equivalent and produced a counterexample, a long execution sequence (178 transitions) accepted by the ring protocol. This sequence clearly violates mutual exclusion, since it ends with:

... $\tau^* . (\text{OPEN } !A3) . \tau^* . (\text{OPEN } !A2)$

A similar result was obtained for the Chang–Roberts' algorithm.

As the diagnostic sequences provided by ALDÉBARAN were much too long to be exploited, we used the EXHIBITOR tool for searching (on-the-fly) the shortest execution

¹⁸ In fact, Chang and Roberts used a different convention than Le Lann, i.e., the station that is elected is the one with the greatest address; however, we keep Le Lann's convention here to maintain consistency in this paper.

¹⁹ In this paper, we focus on Chang–Roberts' core algorithm, and we do not consider the possible variants suggested in the “Startup conditions” and “Concluding comments” sections of [7].

²⁰ Chang–Roberts' algorithm requires $(2n - 1)$ messages in the best case and $n(n + 1)/2$ messages in the worst case. There exist other leader election algorithms [43, 9] with a worst case message bound of $O(n \log n)$.

²¹ In this experiment and the next ones, we assume (presumably without loss of generality) that stations are ordered on the ring by increasing addresses.

sequences starting from the initial state and having the following form:

$$\tau^* . (\text{OPEN } !A_i) . \tau^* . (\text{OPEN } !A_j)$$

where $i \neq j$.

By looking at the minimal scenarios generated by EXHIBITOR, we were able to understand the reason of the error: let S be the station with the smallest address; (1) S emits a first candidature while the token is still circulating; (2) S receives the token and passes it to the next station; (3) S emits a second candidature; (4) S receives the first candidature that completed its circular trip; (5) S believes that it is elected and generates a new token, whereas another token is already circulating.

It is worth noticing that this problem must also occur with unreliable links, since all execution sequences observed with reliable links are also possible with unreliable ones (this was also verified automatically).

In order to understand why the algorithms did not work as we expected initially, we read Le Lann's article again and realized that the state machine expressing the behaviour of the station had to be completed with an additional constraint, the so-called *precedence rule*. [36] formulates this constraint as follows: "a station which has generated a candidature and receives the token before its own candidature has completed a period must remove this candidature from the ring".

Obviously, this precedence rule is not simple to implement, since the ring topology does not allow a given station to remove directly messages which are detained by remote stations. [36] does not explain how to combine the precedence rule and the state machines. [7] takes Le Lann's algorithm as it is and does not evoke this problem. In the next sections, we investigate how the election algorithms could be modified to take into account the precedence rule.

5.3. A first implementation of the precedence rule

A simple solution for enforcing the precedence rule is to make sure that the ring never contains two different candidatures issued by the same station at the same time. In this approach, each station should avoid emitting several candidatures during the same election.

This can be implemented by adding a boolean variable "N" to each station. This variable is *true* when a candidature of this station circulates on the ring; initialized to *false*, it is set to *true* when a candidature is emitted, and to *false* when a new election is started. For similar reasons, candidatures should only be emitted in state α . These modifications to Le Lann's algorithm lead to the following LOTOS description:

```

1 process STATION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, INIT:BOOL):noexit :=
2   ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, false)
3 endproc

4 process ELECTION [OPEN,CLOSE,PRED,SUCC] (Ai:ADDR, S:STATE, N:BOOL):noexit :=
5   [(S == ALPHA) and not (N)] ->
```

```

6     SUCC !CLAIM !Ai; (* timeout *)
7     ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, BETA, true)
8     []
9     PRED !TOKEN;
10    PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, N)
11    []
12    PRED !CLAIM ?Aj:ADDR;
13    (
14    [Ai < Aj] ->
15    SUCC !CLAIM !Aj;
16    ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S, N)
17    []
18    [Ai > Aj] ->
19    SUCC !CLAIM !Aj;
20    (
21    [S == BETA] ->
22    ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, GAMMA, N)
23    []
24    [S <> BETA] ->
25    ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S, N)
26    )
27    []
28    [Ai == Aj] ->
29    (
30    [S == BETA] ->
31    PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, false)
32    []
33    [S <> BETA] ->
34    ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, false)
35    )
36    )
37 endproc

38 process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, N:BOOL) : noexit :=
39 OPEN !Ai;
40 CLOSE !Ai;
41 SUCC !TOKEN;
42 ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, N)
43 []
44 SUCC !TOKEN;
45 ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, N)
46 endproc

```

Similarly, a modified version of Chang–Roberts’ algorithm can be obtained by removing line 15 of the above LOTOS text.

Using CÆSAR and ALDÉBARAN, we verified that a ring containing three modified versions of Le Lann’s (resp Chang–Roberts’) stations and three reliable links is branching equivalent to the service graph (see Appendix A under references #5 and #6).

When replacing the reliable links with *semi-reliable* links (i.e., links that may lose tokens but not candidatures), we also verified that branching equivalence was preserved (see Appendix A under references #7 and #8).

However, when replacing the semi-reliable links with *fully unreliable* links (i.e., links that may lose tokens as well as candidatures), we observed that branching

equivalence was not satisfied (see Appendix A under references #9 and #10). In both cases, ALDÉBARAN provided execution sequences (having 15 transitions and 38 transitions respectively) leading to deadlock states. As these execution sequences were certainly not minimal, we used the TERMINATOR tool to search for the shortest execution sequences leading to deadlock, which produced the following result:

(SUCC3 !CLAIM !A3) . (SUCC2 !CLAIM !A2) . (SUCC1 !CLAIM !A1) . *deadlock*

The deadlock scenario is the following: initially, each station issues a candidature that is immediately lost by the corresponding link; then, each station is waiting for a receipt and cannot issue a new candidature, because one has already been emitted.

However, we verified using ALDÉBARAN that the ring is safety equivalent to the service graph, which means that, even if the whole system can get into deadlock, mutual exclusion is never violated.

We therefore conclude that the modified algorithms described in this section are correct with respect to token losses. However, in presence of candidature losses, they preserve mutual exclusion, but not equal opportunity. In the next section, we will take into account both token and candidature losses.

5.4. A second implementation of the precedence rule

Quite logically, we now seek for an algorithm that would be “fully tolerant”, i.e., would be robust enough to recover from all kind of losses from the links. Such an algorithm was suggested to us by Laurent Gauthier, a former student of the first author, and we will hereafter consider an adaptation of Gauthier’s algorithm [18].

The key idea of this algorithm is to stamp all candidatures with an additional field indicating in which election these candidatures have been issued. This allows each station to distinguish between candidatures belonging to different elections (thus avoiding the mutual exclusion problem mentioned in Section 5.2) and to issue several candidatures during the same election (thus avoiding the deadlock problem mentioned in Section 5.3).

Instead of using an integer counter to number each election round, a simple bit is used, as only two different values are sufficient. The resulting algorithms can be thought as a combination of a leader election protocol and an alternating bit protocol.

Each station has a boolean variable “B”, the value of which denotes the current election round. The value of “B” can be either *true* or *false* initially; it is complemented every time the station leaves the privilege state and transmits the token to the next station. Therefore, a new election round starts after each token receipt or token generation.

When a station receives one of its own candidatures, it determines whether this candidature was generated during the current election round, by checking whether the control bit attached to the candidature is equal to the current value of “B”. If so, the station enters in privileged state and generates a new token. If not, the candidature is silently discarded.

Another boolean variable “C” is added to each station. This variable is set to *true* initially, and every time the token is received. Its value remains *true* as long as the station believes it can win the current election round, i.e., as long as it does not receive a candidature emitted by a station having a smaller address. A station can only candidate when “C” is true.

```

1 process STATION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, INIT:BOOL) : noexit :=
2   ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, true)
3 endproc

4 process ELECTION [OPEN,CLOSE,PRED,SUCC] (Ai:ADDR, C:BOOL, B:BOOL):noexit :=
5   [C] ->
6     SUCC !CLAIM !Ai !B;
7     ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, B)
8   []
9   PRED !TOKEN;
10  PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, B)
11  []
12  PRED !CLAIM ?Aj:ADDR ?BO:BOOL;
13  (
14    [Ai < Aj] ->
15      SUCC !CLAIM !Aj !BO;
16      ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, C, B)
17    []
18    [Ai > Aj] ->
19      SUCC !CLAIM !Aj !BO;
20      ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, false, B)
21    []
22    [Ai == Aj] ->
23      (
24        [(BO <> B) or not (C)] ->
25          ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, C, B)
26        []
27        [(BO == B) and C] ->
28          PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, B)
29      )
30  )
31 endproc

32 process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, B:BOOL): noexit :=
33   OPEN !Ai;
34   CLOSE !Ai;
35   SUCC !TOKEN;
36   ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, not (B))
37  []
38  SUCC !TOKEN;
39  ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, not (B))
40 endproc

```

The above algorithm can be modified according to Chang–Roberts’ suggestion, in order to reduce the number of circulating messages. This can be done by removing line 15. Under this modification, the following invariant holds: when a station receives one of its own candidatures (i.e., when $A_i = A_j$), the value of its variable C must

be *true* because only candidatures with the smallest address can do a complete round without being filtered by other stations; thus, the station receiving such a candidature has necessarily the smallest address and was never able to receive a candidature with a smaller address. Therefore, lines 24 and 27 of the above algorithm can be replaced with “[B0 <> B] ->” and “[B0 == B] ->”, respectively.

We used CÆSAR and ALDÉBARAN to verify a ring containing three modified versions of Le Lann’s (resp. Chang–Roberts’) stations and three “fully unreliable” links. We first tried to use CÆSAR to generate directly the corresponding graph, but it happened to be very large, so we switched to a compositional approach, splitting the ring into six communicating processes: three stations and three links. We used CÆSAR and ALDÉBARAN to generate and minimize (modulo strong²² equivalence) the graph of each station and each link separately. Then we used ALDÉBARAN BDD-based algorithms [13] to prove on-the-fly that the ring was branching equivalent to the service graph (see Appendix A under references #11 and #12).

We also studied variants of both algorithms in which line 5 was removed. Using CÆSAR and ALDÉBARAN, we found that removing the guard “[C] ->” affects the correctness of the Le Lann variant: the mutual exclusion property is violated (see Appendix A under reference #13). However, the same modification preserves the correctness of the Chang–Roberts’ variant, although it leads to larger graphs because more candidatures can be emitted (see Appendix A under reference #14). Variable “C” could even be completely removed from the Chang–Roberts’ variant: this variable is not required for the correctness of the algorithm, it is only useful for performance reasons (it reduces the number of exchanged messages).

6. Token-passing with unreliable links and unreliable stations

It is easy to check that, in all above algorithms, new tokens are always generated by the same station, the one with the smallest address. This is a consequence of our assumption about station reliability. We now consider a more elaborate failure model, which cumulates both failures from the links and crashes from the stations.

6.1. Station crashes

The crash model we consider here is the *fail-silent behaviour* mentioned in [36, 7]: whenever a station crashes, it stops forever any message emission. This simple assumption needs to be somehow adapted to our context:

- When a station crashes, the ring topology should not be broken. In [36] this problem is solved by assuming the existence of a ring reconfiguration algorithm. In the token

²² Although branching equivalence would seem natural in this context, we preferred strong equivalence because these equivalences coincide when the graph to minimize has no τ -transitions, which is the case for our stations and links, and because the Paige–Tarjan algorithm for strong equivalence [42] implemented in ALDÉBARAN is more efficient.

ring protocol [31], each station is connected to the ring using a *coupler*, which remains functioning even if the station itself crashes.

We will therefore assume that a crashed station only stops *spontaneous* message emission, but continues to accept messages from its previous station and to deliver these messages to the next station. Thus, crashed stations become “transparent” to other stations on the network, without altering the communication network.

Note that the case in which a crashed station can randomly loose messages is already covered if we consider the combination of a potentially crashing station with its unreliable output link. This leads to a failure model much more severe than the one of [36] and [7], in which links are supposed to be reliable.

- If a station crashes during an election, its candidature might continue to circulate for a while on the ring. With the Chang–Roberts’ algorithm in particular, if the station with the smallest address crashes, its candidature might remain forever, without being filtered, preventing other stations from being elected.

This problem is acknowledged in [7], but no solution is proposed. For the same purpose, [36] suggests that other stations (e.g., the last elected station) could remove from the ring the candidatures emitted by crashed stations (using, for instance, mutual help and mutual suspicion algorithms). We adopt here a simpler approach, assuming that (the coupler of) a crashed station has to filter all candidatures generated by the station before crashing.

- When a station crashes, the other stations are not directly informed about the incident: this would be an unrealistic failure model in a distributed system framework. However, for verification purpose, we must be able to observe crashes (this will be justified in Section 6.2) and we introduce a new “CRASH ! A_i ” event indicating that station S_i has just crashed.

Under these assumptions, the description in LOTOS of a potentially crashing station can be the following:

```

1 process STATION [OPEN,CLOSE,CRASH, PRED, SUCC] (Ai:ADDR, INIT:BOOL):noexit :=
2   ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, true)
3   [>
4     CRASH !Ai;
5     FAIL [PRED, SUCC] (Ai)
6 endproc
```

where:

- the call to process “ELECTION” expresses the normal functioning mode of the station;
- the LOTOS operator “[>” (called the *disabling* operator) expresses that in any state of its normal behaviour, the station can be disrupted by a “CRASH ! A_i ” action, then entering into a fail-silent behaviour mode;
- the fail-silent behaviour mode is defined by the process “FAIL” which accepts both tokens and candidatures, filters candidatures stamped with address A_i , and transmits tokens and candidatures stamped with an address A_j different from A_i .

6.2. Service with station crashes

The simple definition of the service given in Section 4.1 must be modified to take into account the possibility of station crashes. There are some fundamental changes:

- A station S_i might crash while it is accessing the shared resource. This modifies the mutual exclusion property, since it becomes possible to have an execution sequence of the form:

$$\tau^*. (\text{OPEN } !A_i). \tau^*. (\text{CRASH } !A_i) . \tau^* . (\text{OPEN } !A_j) \dots$$

in which there is no “CLOSE $!A_i$ ” occurring between events “OPEN $!A_i$ ” and “OPEN $!A_j$ ”. This is the reason why we chose to model crashes as visible events, not as τ actions, in order to be able to distinguish such situations from “actual” violations of mutual exclusion.

- A station S_j may crash while a station S_i is accessing the shared resource. This allows execution sequences of the form:

$$\tau^* . (\text{OPEN } !A_i) . \tau^* . (\text{CRASH } !A_j) . \tau^* . (\text{CLOSE } !A_i) \dots$$

However, such sequences are only allowed if $i \neq j$.

The corresponding service graph is too complex to be modelled by hand, even for a limited number of three stations. We found more convenient to express this service more abstractly, as two mutually recursive LOTOS processes. Both processes are parameterized by a variable “E”, which denotes the set of addresses of functioning stations; only those stations S_i whose addresses A_i belong to “E” can perform “OPEN $!A_i$ ”, “CLOSE $!A_i$ ” or “CRASH $!A_i$ ” actions; initially, “E” contains all addresses; when a station crashes, its address is removed from “E”.

```

1 SERVICE [OPEN, CLOSE, CRASH] ({} + A1 + A2 + A3)
2 where
3 process SERVICE [OPEN, CLOSE, CRASH] (E:ADDR_SET) : noexit :=
4   choice Ai:ADDR []
5     (
6       [Ai isin E] ->
7         (
8           OPEN !Ai;
9             SERVICE_BIS [OPEN, CLOSE, CRASH] (E, Ai)
10          []
11          CRASH !Ai;
12            SERVICE [OPEN, CLOSE, CRASH] (E - Ai)
13         )
14      )
15 endproc

16 process SERVICE_BIS [OPEN, CLOSE, CRASH] (E:ADDR_SET, Ai:ADDR) : noexit :=
17   CLOSE !Ai;
18   SERVICE [OPEN, CLOSE, CRASH] (E)
19   []
20   CRASH !Ai;
21   SERVICE [OPEN, CLOSE, CRASH] (E - Ai)

```

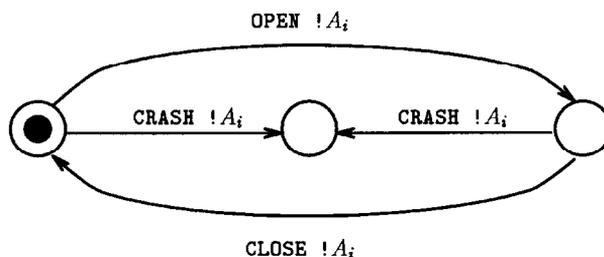


Fig. 3. Graph expressing projection of service with crashes on station S_i .

```

22   []
23   (
24   choice Aj:ADDR []
25   (
26   [(Aj isin E) and (Aj <> Ai)] ->
27   CRASH !Aj;
28   SERVICE_BIS [OPEN, CLOSE, CRASH] (E - Aj, Ai)
29   )
30   )
31 endproc

```

The graph of this service (generated using *CÆSAR* and minimized using *ALDÉBARAN* modulo branching bisimulation) has 20 states and 60 transitions. It is deterministic and has a single deadlock state, reached after successive crashing of all the stations. Also, if we only consider the actions performed by a single station S_i (by hiding the actions of the other stations and reducing the resulting graph using branching bisimulation), we obtain the “projection” graph represented on Fig. 3.

6.3. Protocol with station crashes

We now seek for an algorithm that would be tolerant to both medium losses and station crashes. We consider the algorithm given in Section 5.4 to which we have applied all the aforementioned changes related to the Chang–Roberts’ optimization.

It is clear that, in order to avoid deadlocks, the guard “[C] ->” on line 5 of this algorithm must be suppressed. Otherwise, if the station with the smallest address crashes after having sent candidatures, other stations might be prevented from sending their own candidatures because they previously received a “stronger” candidature from a station which is no longer functioning.

As we already mentioned in Section 5.4, removing the guard “[C] ->” does not affect the correctness of the algorithm in presence of link losses and that variable “C” could be suppressed in such case. Our algorithm is therefore derived from the one listed in Section 5.4 by removing lines 5 and 15, by replacing lines 24 and 27 with “[B0 <> B] ->” and “[B0 == B] ->” respectively, and by removing all occurrences of variable “C”.

Using CÆSAR and ALDÉBARAN, we found that a ring with three of these (potentially crashing) stations and three fully unreliable links was branching equivalent to the service presented in Section 6.2. This verification was carried out using the compositional approach described in Section 5.4 (see Appendix A under reference #15).

One could wonder if modeling station crashes with the “disable” operator of LOTOS has not the undesirable effect of masking potential deadlocks, i.e., a station which is in a deadlock state being able to escape from this situation by crashing, then executing the “FAIL” process. This answer to this difficult question is negative in this particular case: branching equivalence guaranties that the ring behaviour will have the same “branching structure” as the service graph defined in Section 6.2; by looking at the projections of this graph given on Fig. 3, one observes that, when a station S_i can crash, it can always do another action (“OPEN ! A_i ” or “CLOSE ! A_i ”) at the same time; therefore, crashing is never an artefact to escape from deadlock.

7. Conclusions

The distributed leader election problem was originally coined by Le Lann in 1977, for which several algorithms have been proposed during the last seventies and early eighties. But, to our knowledge, it seems that these algorithms have been lacking a formal treatment until very recently. In this article, we have studied several variants of two existing leader election algorithms [36, 7] for stations connected by a unidirectional ring network consisting of reliable or unreliable one-slot buffers. Two related experiments can be found in the recent literature: [14] gives a formal specification and proves the correctness of Dolev–Klawe–Rodesh’s leader election algorithm for stations connected by a reliable, unidirectional ring; [4, 5] give a formal specification and proves the correctness of several distributed leader election algorithms for stations communicating via asynchronous broadcasting facilities. Also, an older article [6] discusses the partial validation of Chang–Roberts’ algorithm using random simulation.

We can draw several conclusions from our case-study. First, we have pointed out that the algorithms proposed in [36, 7] were not complete (the precedence rule was not implemented) and did not guarantee the uniqueness of the elected leader in a token-passing context where several elections can take place in sequence. We have also pointed out that these algorithms did not resist to losses of messages by communication channels. We studied variants of these algorithms and proposed an improved, simpler algorithm that tolerates both link failures and station crashes.

We believe that describing distributed algorithms formally is a fruitful task, since the use of a formal method is likely to reveal ambiguities that would otherwise remain hidden. Such an approach is a considerable improvement over the current practice, often based upon informal (natural language) or semi-formal (state diagrams) notations. We consider that the international standard LOTOS is very appropriate for a concise description of complex distributed systems, as it combines clean theoretical concepts borrowed from process algebras with high-level features suitable for software-engineering. We

found some LOTOS operators very useful, for instance the disabling operator, which is necessary for a compositional description of station crashes.

We have shown that automated verification techniques can be successfully applied to non-trivial problems. In spite of the conciseness of their LOTOS descriptions, leader election algorithms are very complex, if not beyond the limits of human reasoning capabilities. Although this article presents our experiments in a rather ordered way, this was not the case actually: when designing the algorithms, we followed many trial-error-correction cycles and the right solution was never found at the first time.

We based our verification upon the use of bisimulations, by comparing (modulo branching or safety equivalence) the graph of a protocol against the graph of its expected service. We believe that bisimulation technology offers a practical alternative to other verification approaches based upon temporal logics, for instance [14, 4, 5]. This is specially true in the present case-study, where the use of branching bisimulation guarantees mutual exclusion and equal opportunity at the same time, while many temporal logic formulas would otherwise be required to express the same properties. In general, however, it is not easy to determine which properties are to be verified, and how. This article provides a sketch of methodology, which, we believe, could be reused for the specification and verification of other classes of distributed algorithms.

As regards efficiency, Appendix A indicates that the correctness of the algorithms can be checked in a few seconds (less than one minute, in most cases). This demonstrates the ability of the CADP toolbox to deal with complex systems. This was not true in 1988, when we started to study these leader election algorithms. We had to wait until 1992 to perform complete verification, because the tools were not able to tackle the whole complexity of the system, and did not provide us with enough diagnostic information to understand why verification failed. The former problem was solved by enhancing ALDÉBARAN with on-the-fly, compositional and BDD-based verification techniques, the latter by introducing the OPEN/CÆSAR environment and its on-the-fly analysis tools EXHIBITOR and TERMINATOR.

We should underline the fact that our verifications have been carried out for three stations only. Given the figures of Appendix A, it is likely that our experiments could be extended to a greater number of stations. However, our approach does not allow to prove the correctness for any number n of stations: a token-passing algorithm might very well be correct for some values of n (e.g., when n is a prime number), but not all. The correctness for all n could be established either by hand-written proofs, as in [36, 14, 4, 5], or by a combination of manual and computer-aided techniques, in which automatic demonstrators are used to verify induction hypotheses provided by human experts. However, finding induction hypotheses for leader election algorithms is not easy, due to the circular topology of the ring, and because all stations are not exactly alike (the behaviour of a station depends of its address). This remains a challenging problem for further research.

It is therefore clear that model-based verification techniques cannot replace traditional proofs of correctness. Yet, they can be used as “intelligent debuggers”, which

help to detect design errors earlier, thus increasing the level of confidence in a design and providing useful knowledge about the behaviour of the system. This is more or less the usual approach followed by designers of distributed systems, who often develop prototypes in C or C++ to simulate their designs. We are convinced that a LOTOS-based approach is more effective than the current practice: LOTOS descriptions are significantly shorter than their counterpart in C and several compilers exist that generate C code from LOTOS descriptions. Besides its formal verification capabilities, the CADP toolbox provides interactive simulation facilities (with advanced features such as unlimited backtracking), as well as partial verification techniques that outperform simulation techniques by allowing a much larger coverage of the possible execution sequences.

Acknowledgements

The authors are grateful to Jean-Claude Fernandez, Alain Kerbrat (Verimag, Grenoble, France) and to Roland Groz (CNET Lannion, France) who have experimented other variants of the leader election algorithms, and would like to thank Guy Leduc (University of Liège, Belgium) and the anonymous referees for their insightful comments about this article.

Appendix A. Collected statistics

This appendix contains measures related to the verification of a network of three stations and three links. We assume that the addresses of the stations are ordered as follows: $A_1 < A_2 < A_3$. All the execution times have been obtained on a SparcStation 20 running Solaris 2.4 with 128 megabytes of main memory.

The conventions used are shown in Table 1. There are ten different kinds of stations, five different kinds of links, and four possible verification results.

Table 2 summarizes the measures obtained for enumerative verification: first, the graph corresponding to the network is generated using CÆSAR, then it is compared, modulo branching equivalence, to the service graph of Section 4.1. For each experiment, the table gives: the kind of stations and links involved; the number of states and transitions of the generated graph; the verification result; and the total time needed for the complete verification process (in seconds).

Table 3 summarizes the measures obtained for compositional, on-the-fly verification: first, the graphs of the stations and links are generated separately using CÆSAR and minimized, modulo strong equivalence, using ALDÉBARAN; then ALDÉBARAN is used to compare on-the-fly, modulo branching equivalence, the ring behaviour with respect to the service graph of Section 4.1 (except for experiment #15, where the service graph of Section 6.2 is used instead). For each experiment, the table gives: the kind of stations and links involved; the sizes of all reduced graphs under the form “ $(s_1, t_1) \times (s_2, t_2) \times$

Table 1

Abbreviation	Kind of station	See Section ...
<i>B</i>	basic station	4.4
<i>LL</i>	Le Lann's original algorithm	5.2
<i>CR</i>	Chang–Roberts' original algorithm	5.2
<i>LL</i> ₁	Le Lann's algorithm with first precedence rule	5.3
<i>CR</i> ₁	Chang–Roberts' algorithm with first precedence rule	5.3
<i>LL</i> ₂	Le Lann's algorithm with second precedence rule	5.4
<i>CR</i> ₂	Chang–Roberts' algorithm with second precedence rule	5.4
<i>LL</i> ₃	Same as <i>LL</i> ₂ without the “[C] ->” guard	5.4
<i>CR</i> ₃	Same as <i>CR</i> ₂ without the “C” variable	5.4
<i>F</i>	Same as <i>CR</i> ₃ with station crashes	6.3
Abbreviation	Kind of link	See Section(s) ...
<i>T</i>	Link carrying tokens reliably	4.3
\bar{T}	Link carrying tokens unreliably	5.1
<i>T.C</i>	Link carrying tokens and claims reliably	4.3
$\bar{T}.C$	Links carrying tokens unreliably and claims reliably	5.1 and 5.4
$\bar{T}.\bar{C}$	Fully unreliable link	5.1 and 5.4
Abbreviation	Verification result	
\approx_B	branching equivalence is satisfied	
\approx_S	safety equivalence is satisfied	
$\not\approx_B$ [e]	branching equiv. is not satisfied, with violation of mutual exclusion	
$\not\approx_B$ [d]	branching equiv. is not satisfied, with presence of deadlocks	

Table 2

Reference	Station	Link	States	Trans	Result	Time
#1	<i>B</i>	<i>T</i>	42	52	\approx_B	5
#2	<i>B</i>	\bar{T}	50	72	$\not\approx_B$ [d]	5
#3	<i>LL</i>	<i>T.C</i>	126,577	319,010	$\not\approx_B$ [e]	59
#4	<i>CR</i>	<i>T.C</i>	43,296	115,108	$\not\approx_B$ [e]	28
#5	<i>LL</i> ₁	<i>T.C</i>	16,985	42,423	\approx_B	14
#6	<i>CR</i> ₁	<i>T.C</i>	6,572	14,516	\approx_B	9
#7	<i>LL</i> ₁	$\bar{T}.C$	30,085	77,680	\approx_B	24
#8	<i>CR</i> ₁	$\bar{T}.C$	9,308	21,078	\approx_B	13

Table 3

Reference	Station	Link	Size of communicating processes	States	Trans.	Result	Time
#9	<i>LL</i> ₁	$\bar{T}.\bar{C}$	$(15, 27) \times (14, 26) \times (13, 25) \times (5, 12)^3$	3,759	10,883	$\not\approx_B$ [d] \approx_S	48
#10	<i>CR</i> ₁	$\bar{T}.\bar{C}$	$(9, 21) \times (11, 23) \times (13, 25) \times (5, 12)^3$	1,373	3,908	$\not\approx_B$ [d] \approx_S	29
#11	<i>LL</i> ₂	$\bar{T}.\bar{C}$	$(16, 32) \times (22, 50) \times (18, 46) \times (8, 21)^3$	81,888	250,944	\approx_B	45
#12	<i>CR</i> ₂	$\bar{T}.\bar{C}$	$(8, 24) \times (14, 42) \times (18, 46) \times (8, 21)^3$	9,568	32,808	\approx_B	31
#13	<i>LL</i> ₃	$\bar{T}.\bar{C}$	$(16, 32) \times (22, 52) \times (18, 48) \times (8, 21)^3$	625,440	1,795,200	$\not\approx_B$ [e]	270
#14	<i>CR</i> ₃	$\bar{T}.\bar{C}$	$(8, 24) \times (12, 28) \times (16, 32) \times (8, 21)^3$	10,512	33,896	\approx_B	38
#15	<i>F</i>	$\bar{T}.\bar{C}$	$(14, 44) \times (18, 52) \times (22, 60) \times (18, 96)^3$	162,995	609,297	\approx_B	69

$(s_3, t_3) \times (s_L, t_L)^{3^n}$ where s_i and t_i denote the number of states and transitions of the reduced graph for station S_i , and s_L and t_L denote the number of states and transitions of the reduced graph for the links (all the links are the same); the number of states and transitions of the product graph (obtained using BDD computations); the verification result; and the total time necessary for the complete verification process (in seconds).

References

- [1] M. Abadi and L. Lamport, The existence of refinement mappings, *Theoret. comput. Sci.* **82**(2) (1991) 253–284.
- [2] T. Bolognesi and E. Brinksma, Introduction to the ISO specification language LOTOS, *Comput. Networks ISDN Systems* **14** (1988) 25–29.
- [3] A. Bouajjani, J.-Cl. Fernandez, S. Graf, C. Rodriguez and J. Sifakis, Safety for branching time semantics, in: *Proc. 18th ICALP*, Berlin (Springer, Berlin, 1991).
- [4] J.J. Brunekreef, J.-P. Katoen, R.L.C. Koymans and S. Mauw, Algebraic specification of dynamic Leader Election Protocols in broadcast networks, in: A. Ponse, C. Verhoef and S.F.M. van Vlijmen, eds., *Proc. Workshop on Algebra of Communicating Processes ACP'94*, Workshops in Computing (Springer, Berlin, 1995), 338–357. Also available as Technical Report P9324, University of Amsterdam.
- [5] J.J. Brunekreef and J.-P. Katoen, R.L.C. Koymans and S. Mauw, Algebraic specification of dynamic leader election protocols in broadcast networks, *Distrib. Comput.* **9**(4) (1996) 157–171.
- [6] A. Cavalli and E. Paul, Exhaustive analysis and simulation for distributed systems, both sides of the same coin, *Distrib. Comput.* **2** (1988).
- [7] E. Chang and R. Roberts, An improved algorithm for decentralized extrema-finding in circular configurations of processes, *Comm. ACM* **22**(5) (1979) 281–283.
- [8] J. de Meer, R. Roth and S. Vuong, Introduction to algebraic specifications based on the language ACT ONE, *Comput. Networks ISDN Systems* **23** (1992) 363–392.
- [9] D. Dolev, M. Klawe and M. Rodeh, An $O(n \log n)$ unidirectional distributed algorithm for extrema finding in a circle, *J. Algorithms* **3** (1982) 245–260.
- [10] J.-C. Fernandez, An implementation of an efficient algorithm for bisimulation equivalence, *Sci. Comput. Programming* **13** (1990) 219–236.
- [11] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier and M. Sighireanu, CADP (CÆSAR/ALDEBARAN Development Package): A protocol validation and verification toolbox, in: R. Alur and T.A. Henzinger, eds., *Proc. 8th Conf. on Computer-Aided Verification* (New Brunswick, NJ, USA), Lecture Notes in Computer Science, Vol. 1102 (Springer, Berlin, 1996) 437–440.
- [12] J.-C. Fernandez, H. Garavel, L. Mounier, A. Rasse, C. Rodriguez and J. Sifakis, A toolbox for the verification of LOTOS programs, in: L.A. Clarke, ed., *Proc. 14th Internat. Conf. on Software Engineering ICSE'14*, Melbourne, Australia (ACM, New York, 1992) 246–259.
- [13] J.-C. Fernandez, A. Kerbrat and L. Mounier, Symbolic equivalence checking, in: C. Courcoubetis, ed., *Proc. 5th Workshop on Computer-Aided Verification*, Heraklion, Greece, Lecture Notes in Computer Science, Vol. 697 (Springer, Berlin, 1993).
- [14] L.A. Fredlund, J.F. Groote and H.P. Korver, Formal verification of a leader election protocol in process algebra, in: *Proc. Workshop on Algebra of Communicating Processes ACP'95* (1995) 285–308; Computing Science Reports, Report 95-14, Eindhoven University of Technology; Also Technical Report R95-01, Swedish Institute of Computer Science.
- [15] H. Garavel, Compilation of LOTOS abstract data types, in: S.T. Vuong, ed., *Proc. 2nd Internat. Conf. on Formal Description Techniques FORTE'89*, Vancouver B.C., Canada (North-Holland, Amsterdam, 1989) 147–162.
- [16] H. Garavel and J. Sifakis, Compilation and verification of LOTOS specifications, in: L. Logrippo, R.L. Probert and H. Ural, eds., *Proc. 10th Internat. Symp. on Protocol Specification, Testing and Verification*, Ottawa, Canada (IFIP, North-Holland, Amsterdam, 1990) 379–394.
- [17] H. Garavel and P. Turlier, CÆSAR.ADT: un compilateur pour les types abstraits algébriques du langage LOTOS, in: R. Dssouli and G.v. Bochmann, eds., *Actes du Colloque Francophone pour l'Ingénierie des Protocoles CFIP'93*, Montréal, Canada (1993).

- [18] L. Gauthier, Private communication, 1992.
- [19] J.F. Groote and F. Vaandrager, An efficient algorithm for branching bisimulation and stuttering equivalence, in: M.S. Patterson, ed., *Proc. 17th ICALP*, Warwick, Lecture Notes in Computer Science, Vol. 443 (Springer, Berlin, 1990) 626–638.
- [20] C.A.R. Hoare, *Communicating Sequential Processes* (Prentice-Hall, Englewood Cliffs, NJ, 1985).
- [21] ISO/IEC, File transfer, access and management, Internat. Standards 8571-*, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Genève (1988).
- [22] ISO/IEC, LOTOS – A formal description technique based on the temporal ordering of observational behaviour, International Standard 8807, International Organization for Standardization – Information Processing Systems – Open Systems Interconnection, Genève (1988).
- [23] ISO/IEC, LOTOS description of the session protocol, Technical Report 9572, Internat. Organization for Standardization – Open Systems Interconnection, Genève (1989).
- [24] ISO/IEC, LOTOS description of the session service, Technical Report 9571, Internat. Organization for Standardization – Open Systems Interconnection, Genève (1989).
- [25] ISO/IEC, Local area networks – Part 4: Token-passing bus access method and physical layer specifications, Internat. Standard 8802-4, Internat. Organization for Standardization – Information Processing Systems, Genève (1990).
- [26] ISO/IEC, Approved algorithms for message authentication – Part 2: Message authenticator algorithm, Internat. Standard 8731-2, Internat. Organization for Standardization – Banking, Genève (1992).
- [27] ISO/IEC, Distributed transaction processing – Part 3: protocol specification. Internat. Standard 10026-3, Internat. Organization for Standardization – Information Technology – Open Systems Interconnection, Genève (1992).
- [28] ISO/IEC, Formal description of ISO 8072 in LOTOS, Tech. Report 10023, Internat. Organization for Standardization – Telecommunications and Information Exchange between Systems, Genève (1992).
- [29] ISO/IEC, Formal description of ISO 8073 (Classes 0, 1, 2, 3) in LOTOS, Technical Report 10024, Internat. Organization for Standardization – Telecommunications and Information Exchange between Systems, Genève (1992).
- [30] ISO/IEC, Fibre distributed data interface (FDDI), Internat. Standards 9314-*, Internat. Organization for Standardization – Information Processing Systems, Genève (1989–1995).
- [31] ISO/IEC, Local and metropolitan area networks – Part 5: token ring access method and physical layer specifications, Internat. Standard 8802-5, Internat. Organization for Standardization – Information Processing Systems, Genève (1995).
- [32] ISO/IEC, LOTOS description of the CCR protocol, Technical Report 11590, Internat. Organization for Standardization – Open Systems Interconnection, Genève (1995).
- [33] ISO/IEC, LOTOS description of the CCR Service, Technical Report 11589, Internat. Organization for Standardization – Open Systems Interconnection, Genève (1995).
- [34] R. Lai and A. Lo, An analysis of the ISO FTAM basic file protocol specified in LOTOS, *Austral. Comput. J.* **27** (1995) 1–7.
- [35] L. Lamport, Proving the correctness of multiprocess programs, *IEEE Trans. Software Eng.* **3** (1977) 125–143.
- [36] G. Le Lann, Distributed systems – towards a formal approach, in: B. Gilchrist, ed., *Inform. Process.* **77** (IFIP, North-Holland, Amsterdam, 1977) 155–160.
- [37] R. Milner, *A Calculus of Communicating Systems*, Lecture Notes in Computer Science, Vol. 92 (Springer, Berlin, 1980).
- [38] R. Milner, *Communication and Concurrency* (Prentice-Hall, Englewood Cliffs, NJ, 1989).
- [39] L. Mounier, Méthodes de vérification de spécifications comportementales: étude et mise en œuvre, Thèse de Doctorat, Université Joseph Fourier, Grenoble, 1992.
- [40] H.B. Munster, LOTOS specification of the MAA standard, with an evaluation of LOTOS, NPL Report DITC 191/91, National Physical Laboratory, Teddington, Middlesex, UK, 1991.
- [41] D. Park, Concurrency and automata on infinite sequences, in: P. Deussen, ed., *Theoret. Computer Science*, Lecture Notes in Computer Science, Vol. 104 (Springer, Berlin, 1981) 167–183.
- [42] R. Paige and R.E. Tarjan, Three partition refinement algorithms, *SIAM J. Comput.* **16** (1987) 973–989.
- [43] G.L. Peterson, An $O(n \log n)$ unidirectional algorithm for the circular extrema problem, *ACM Trans. Programming Languages and Systems* **4** (1982) 758–762.
- [44] J.-P. Queille and J. Sifakis, Fairness and related properties in transition systems – a temporal logic to deal with fairness, *Acta Inform.* **19** (1983) 195–220.

- [45] A. Tannenbaum, *Computer Networks* (Prentice-Hall, Engelwood Cliffs, NJ, 1989).
- [46] K.J. Turner, ed., *Using Formal Description Techniques – An Introduction to ESTELLE, LOTOS, and SDL* (Wiley, New York, 1993).
- [47] R.J. van Glabbeek and W.P. Weijland, Branching-time and abstraction in bisimulation semantics (extended abstract), CS R8911, Centrum voor Wiskunde en Informatica, Amsterdam, 1989; Also in: *Proc. IFIP 11th World Computer Congress*, San Francisco (1989).
- [48] C.H. West, A general technique for communication protocol validation, *IBM J. Res. Development* (1978) 393–404.