INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Specification and Verification of various Distributed Leader Election Algorithms for Unidirectional Ring Networks

Hubert Garavel, Laurent Mounier

N° 2986

Septembre 1996

THÈME 1

Rapport de recherche

ISSN 0249-6399 ISRN INRIA/RR--2986--FR+ENG

Specification and Verification of various Distributed Leader Election Algorithms for Unidirectional Ring Networks

Hubert Garavel*, Laurent Mounier†

Thème 1 — Réseaux et systèmes
Projet Spectre

Rapport de recherche n° 2986 — Septembre 1996 — 32 pages

Abstract: This report 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 this class of algorithms in presence of unreliable communication medium and/or unreliable machines. We suggest 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. Model-checking and bisimulation techniques allow the verification of these non-trivial algorithms to be carried out automatically.

Key-words: Bisimulations, Distributed Algorithms, Formal Description Techniques, Formal Methods, Labelled Transition Systems, Leader Election, Lotos, Model-checking, Process Algebras, Protocol Engineering, Specification, Token Ring, Validation, Verification.

Short version of this report in *Science of Computer Programming*, special issue on Industrially Relevant Applications of Formal Analysis Techniques, Jan-Friso Groote and Martin Rem, editors, 1996.

* E-mail: `Hubert.Garavel@imag.fr`
† Inria Rhône-Alpes / Verimag, Miniparc-Zirst, rue Lavoisier, F-38330 Montbonnot Saint-Martin, France. E-mail: `Laurent.Mounier@imag.fr`

Spécification et vérification de plusieurs algorithmes distribués d'élection sur un réseau en anneau unidirectionnel

Résumé : Ce rapport présente la spécification et la vérification formelle d'algorithmes distribués d'élection sur un réseau en anneau unidirectionnel.

A partir d'un algorithme proposé par Le Lann en 1977 et de sa variante proposée par Chang et Roberts en 1979, on étudie la robustesse de cette classe d'algorithmes en présence de canaux de communication et/ou de machines non fiables. Nous suggérons différentes améliorations de ces algorithmes afin d'obtenir un protocole résistant à ces pannes.

Ces algorithmes ont été formellement décrits à l'aide du langage de spécification LOTOS normalisé par l'ISO et vérifiés (pour un nombre fixé de machines) grâce à la boîte à outils CADP (CÆSAR/ALDÉBARAN). En utilisant des techniques fondées sur la génération exhaustive et les relations de bisimulation, la vérification de ces algorithmes non-triviaux peut être effectuée automatiquement.

Mots-clés : Algèbres de processus, Algorithmes distribués, Bisimulations, Election, Ingénierie des protocoles, LOTOS, Méthodes formelles, Réseau à jeton, Réseau en anneau, Spécification, Systèmes de transitions étiquetées, Techniques de description formelle, Validation, Vérification.

1 Introduction

Since the early attempts of automatic protocol validation carried out by C. West [Wes78], 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.

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 report 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 G. Le Lann [Lan77] and its variant proposed by E. Chang and R. Roberts [CR79], 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 report 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 LOTOS[1] 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 SEDOS[2] project and standardized by ISO[3] in 1988 [ISO88b]. Several tutorials for LOTOS are available, e.g. [BB88, Tur93].

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. LOTOS features two clearly separated parts:

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 [EM85, dMRV92].

In this approach, data structures are described by *sorts*, which represent value domains, and *operations*, which are mathematical functions defined on these domains. The meaning of operations is defined by algebraic *equations*. Sorts, operations, and equations are grouped in modules

[1] *Language Of Temporal Ordering Specification*
[2] *Software Environment for the Design of Open Distributed Systems*, ESPRIT project 410
[3] *International Organization for Standardization*

called *types*, which can be combined together using importation, renaming, parametrization, and actualization. The underlying semantics is that of initial algebras [EM85].

The control part of LOTOS is based on the process algebra approach for concurrency, and appears to combine the best features of CCS [Mil80, Mil89] and CSP [Hoa85].

LOTOS relies on a small set of basic operators, which represent primitive concepts of concurrent systems (sequential composition, non-deterministic choice, guard, parallel composition, rendezvous, etc.) These operators are used to build algebraic terms that describe the behaviour of concurrent systems; the approach is compositional, since complex behaviours can be obtained by combining elementary ones.

As for most process algebras, the semantics of LOTOS is formally defined in terms of *Labelled Transition Systems* (hereafter called *graphs*) [Par81, Mil89, ISO88b], i.e. directed graphs whose vertices denote the global *states* of the system and whose edges correspond to the evolutions (*transitions*) of the system.

LOTOS has been applied to describe complex systems formally, among which: the service and protocols for the OSI transport and session layers [ISO89b, ISO89a, ISO92c, ISO92d], the CCR[4] service and protocol [ISO95d, ISO95c], OSI TP[5] [ISO92b, Annex H], MAA[6] [ISO92a, Mun91], FTAM[7] basic file protocol [ISO88a, LL95], etc. It has been mostly used to describe software systems, although it can also be used for asynchronous hardware description [CGM+96].

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. For this case-study, we used the CADP toolbox, which provides state-of-the-art verification features.

Since 1993, a revision of the LOTOS standard has been undertaken within ISO. This on-going activity should give birth to a new version of LOTOS, named E-LOTOS (for *Extended LOTOS*), which is expected to introduce simpler features (especially for data types) and increased expressiveness (notably by adding quantitative time to the language). However, this new language being not available at the time of this case-study, we based our work on the existing standard LOTOS.

3 Description of the verification tools used

The CADP[8] toolbox [FGM+92, FGK+96, Gar96] 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.

[4] *Commitment, Concurrency, and Recovery*
[5] *Distributed Transaction Processing*
[6] *Message Authentication Algorithm*
[7] *File Transfer, Access and Management*
[8] CÆSAR/ALDÉBARAN Development Package

- 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 [GS90] and CÆSAR.ADT [Gar89, GT93] are compilers able to translate a LOTOS program into a finite state graph describing its exhaustive behaviour. This graph can be either *explicitly* represented as a 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 [Fer90, FKM93] is a verification tool able either to compare or to minimize graphs with respect to (bi)simulation relations [Par81, Mil80]. 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.

4 Token-passing with reliable links / 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, ..., 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 OSI[9] 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 property[10] [Lam77, AL91]. 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 stations. Equal opportunity is a liveness property[11] [Lam77, AL91]. In particular, under the usual fairness assumption given in [QS83],

[9] ISO standards for Open System Interconnection

[10] 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.

[11] Liveness properties assert that "something good eventually happens".

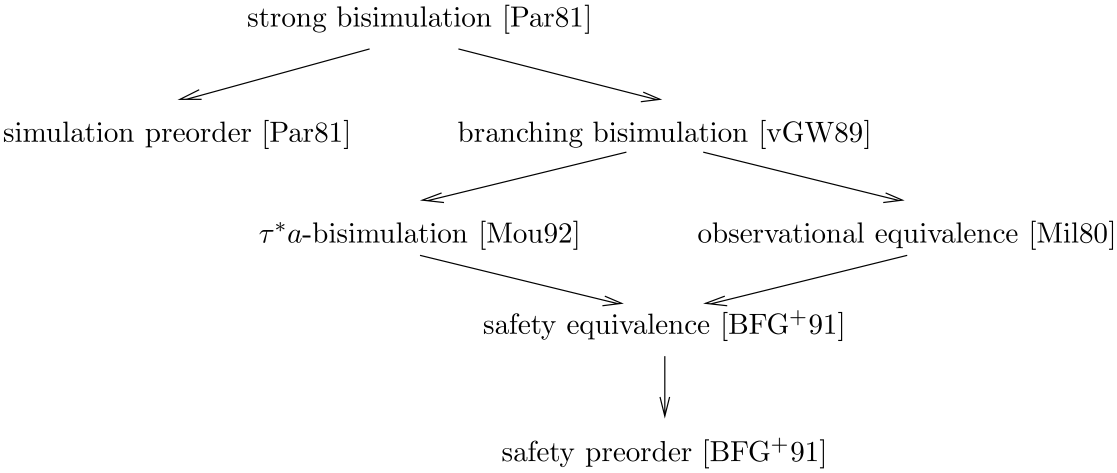

Figure 1: Lattice of the relations currently implemented in ALDÉBARAN

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).

Many relations, especially *bisimulations*, have been defined in the literature. A tentative classification of bisimulation can be found in [Mou92]; Figure 1 shows the lattice of the bisimulation relations which are currently implemented in the 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 [BFG+91] which preserves all safety properties. The graph on the left of Figure 2 is the minimal graph (in number of states) expressing mutual exclusion for three stations with respect to safety equivalence[12].

- 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 by these two relations). Both remaining candidates, observational equivalence and branching bisimulation[13] preserve deadlocks. A first way to express this property consists in hiding, in the protocol graph, all

[12] More precisely, any graph smaller than this graph with respect to the safety preorder satisfies mutual exclusion
[13] Strong equivalence is not useful for verifying graphs which contain invisible (i.e., τ) actions

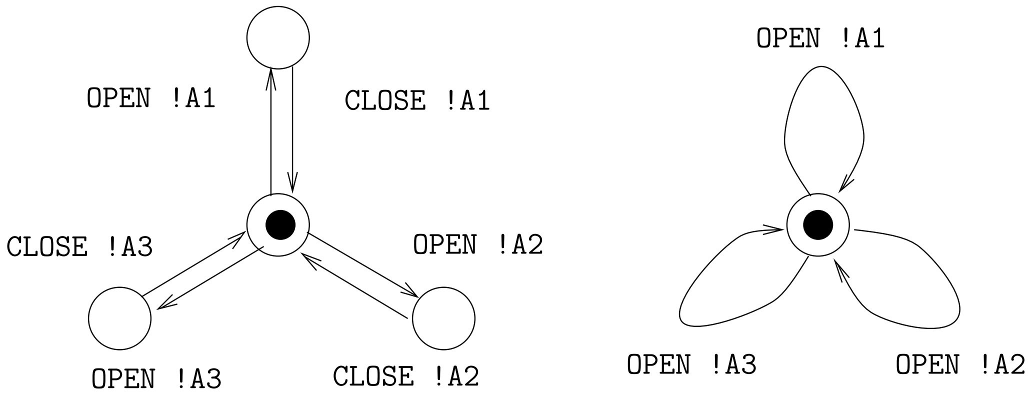

Figure 2: Graphs expressing mutual exclusion and equal opportunity

events but "`OPEN !`A_i" and to compare the resulting graph to the graph given on the right of Figure 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 Figure 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 Figure 2 satisfies both mutual exclusion and equal opportunity[14].

The graph on the left of Figure 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).

```
process SERVICE [OPEN, CLOSE] : noexit :=
   choice Ai:ADDR []
      (
      OPEN !Ai;
         CLOSE !Ai;
            SERVICE [OPEN, CLOSE]
      )
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

[14]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 [Mou92]; moreover, the algorithms for branching bisimulation [GV90, FKM93] are more efficient.

the physical layer of the OSI model (see, e.g., [Tan89]) 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 [ISO95b]) or a logical ring, build upon a tree-like physical network (as in the *token bus* protocol [ISO90]). Some other protocols, such as FDDI[15] [ISO95a], also rely on a more complex, double-ring topology.

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`", having the following profiles:

```
process STATION [OPEN, CLOSE, PRED, SUCC] (A:ADDR, INIT:BOOL) : noexit :=
   ...
endproc
process LINK [INPUT, OUTPUT] : noexit :=
   ...
endproc
```

The "`STATION`" process has four gate parameters (enclosed in brackets) and two value parameters (enclosed in parentheses): the gates "`OPEN`" and "`CLOSE`" connect the station to the shared resource; the gates "`PRED`" and "`SUCC`" connect the station to its input and output links; the parameter "`A`" denotes the address of the station ; the parameter "`INIT`" is *true* iff the station owns the token initially (a single station should have this parameter set to *true*; we assume that it is S_1). The "`LINK`" process has two gate parameters, "`INPUT`" and "`OUTPUT`", which connect the link to its neighbour stations.

The overall architecture of this ring is depicted on Figure 3. To specify the connections between processes, we use six actual gates noted "$\mathtt{PRED}_i$" and "$\mathtt{SUCC}_i$", which instanciate the formal gate parameters "`PRED`" and "`SUCC`" of the stations and the formal gate parameters "`INPUT`" and "`OUTPUT`" of the links. Station S_i receives messages from its input link using gate "$\mathtt{PRED}_i$" and sends messages to its output link using gate "$\mathtt{SUCC}_i$". We instantiate all formal gate parameters "`OPEN`" (*resp.* "`CLOSE`") with a single actual gate also noted "`OPEN`" (*resp.* "`CLOSE`"), in order to express that all the stations share the same access to the resource.

There are several possible ways to describe the ring architecture in LOTOS. The simplest one, listed below, is based upon the following observations: the stations execute concurrently and do not synchronize directly with each other (therefore, they can be composed with the parallel composition operator "`|||`"); the same observation holds for links; the group of stations and the group of links

[15] *Fibre Distributed Data Interface*

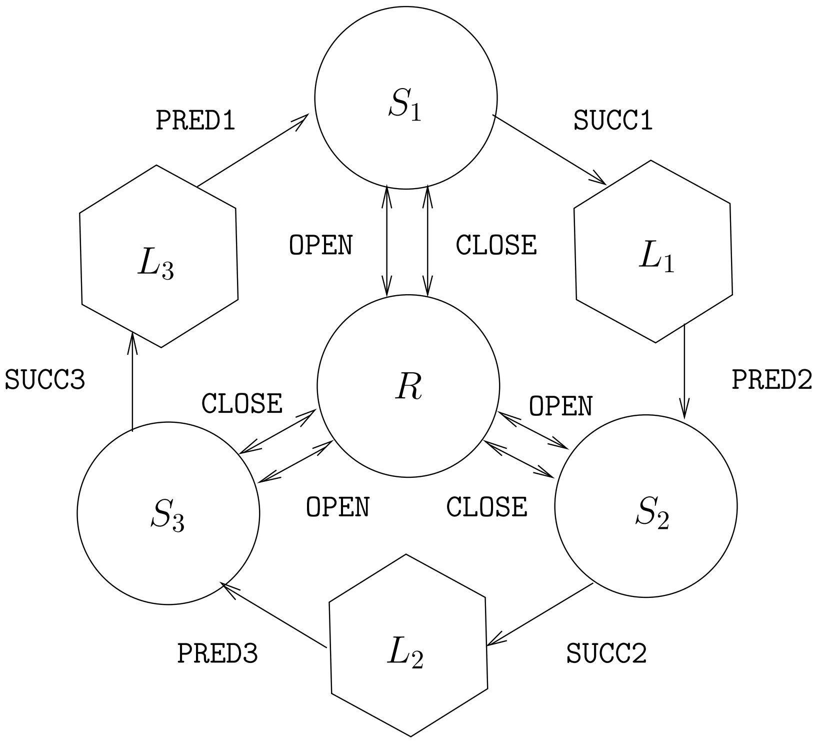

Figure 3: Ring with three stations S_i and three links L_i

synchronize together on all gates "`PRED`$_i$" and "`SUCC`$_i$" (this is expressed using the parallel operator "`|[...]|`"); to verify service properties, we only want to observe "`OPEN`" and "`CLOSE`" gates and we therefore abstract all gates "`PRED`$_i$" and "`SUCC`$_i$" using the "**hide**" operator:

```
hide PRED1, SUCC1, PRED2, SUCC2, PRED3, SUCC3 in
   (
      (
      STATION [OPEN, CLOSE, PRED1, SUCC1] (A1, true)
      |||
      STATION [OPEN, CLOSE, PRED2, SUCC2] (A2, false)
      |||
      STATION [OPEN, CLOSE, PRED3, SUCC3] (A3, false)
      )
   |[PRED1, SUCC1, PRED2, SUCC2, PRED3, SUCC3]|
      (
      LINK [SUCC1, PRED2]
      |||
      LINK [SUCC2, PRED3]
      |||
      LINK [SUCC3, PRED1]
      )
   )
```

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 Lotos description for the reliable link:

```
process LINK [INPUT, OUTPUT] : noexit :=
   INPUT !TOKEN;
      OUTPUT !TOKEN;
         LINK [INPUT, OUTPUT]
endproc
```

In Section 5, we will consider another kind of messages: *candidatures*, which have the form "`!CLAIM !`A_i", where "`CLAIM`" is a constant of the same enumerated type as "`TOKEN`", and where A_i is the address of a station. With two kinds of messages (tokens and candidatures), the behaviour of the reliable link is described by the following Lotos process (where the "`?`" notation means receipt of a value and is used to specify that the address transmitted is equal to the one received, and where the "`[]`" operator denotes non-deterministic choice):

```
process LINK [INPUT, OUTPUT] : noexit :=
   INPUT !TOKEN;
      OUTPUT !TOKEN;
         LINK [INPUT, OUTPUT]
   []
   INPUT !CLAIM ?Ai:ADDR;
      OUTPUT !CLAIM !Ai;
         LINK [INPUT, OUTPUT]
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 is depicted on Figure 4 and 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 looses 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 purposes and will also be present in more sophisticated versions of the

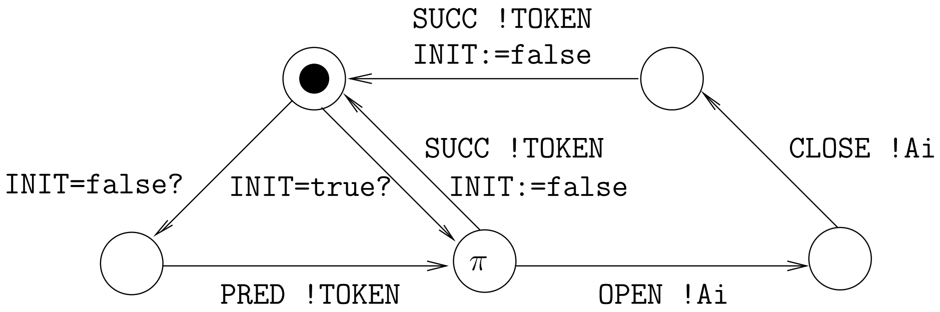

Figure 4: Behaviour graph of the basic station

protocol. The LOTOS operator "->" 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.

```
process STATION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, INIT:BOOL) : noexit :=
   [INIT = true] ->
      PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
   []
   [INIT= false] ->
      PRED !TOKEN;
         PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
endproc

process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR) : noexit :=
   SUCC !TOKEN;
      STATION [OPEN, CLOSE, PRED, SUCC] (Ai, false)
   []
   OPEN !Ai;
      CLOSE !Ai;
         SUCC !TOKEN;
            STATION [OPEN, CLOSE, PRED, SUCC] (Ai, false)
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 Figure 2 using ALDÉBARAN. Quantitative details about this verification can be found in Annex A under reference #1.

The CADP toolbox was also used to study two "pathological" situations in which all stations are initialized with INIT = *false*, or several stations are initialized with INIT = *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 / 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 the token-passing mechanism in the context of Metropolitan Area Networks with possibly unreliable communications. For instance, the token ring standard [ISO95b] 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:

```
process LINK [INPUT, OUTPUT] : noexit :=
   INPUT !TOKEN;
      OUTPUT !TOKEN;
         LINK [INPUT, OUTPUT]
   []
   INPUT !TOKEN;
      LINK [INPUT, OUTPUT]
endproc
```

If we replace reliable links with unreliable ones in a ring consisting of three basic stations, deadlock states can be observed (see Annex A under reference #2).

In Section 5, we will consider *semi-reliable* links, i.e., links that may loose tokens but not candidatures:

```
process LINK [INPUT, OUTPUT] : noexit :=
   INPUT !TOKEN;
      OUTPUT !TOKEN;
         LINK [INPUT, OUTPUT]
   []
   INPUT !TOKEN;
      LINK [INPUT, OUTPUT]
   []
   INPUT !CLAIM ?Ai:ADDR;
      OUTPUT !CLAIM !Ai;
         LINK [INPUT, OUTPUT]
endproc
```

We will also consider *fully unreliable* links, i.e., links that may loose tokens as well as candidatures:

```
process LINK [INPUT, OUTPUT] : noexit :=
   INPUT !TOKEN;
      OUTPUT !TOKEN;
         LINK [INPUT, OUTPUT]
   []
   INPUT !TOKEN;
      LINK [INPUT, OUTPUT]
   []
   INPUT !CLAIM ?Ai:ADDR;
      OUTPUT !CLAIM !Ai;
         LINK [INPUT, OUTPUT]
   []
   INPUT !CLAIM ?Ai:ADDR;
      LINK [INPUT, OUTPUT]
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 [ISO95b]. 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 refered 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 exists 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 [Lan77]. 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[16]. 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.

[16] 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".

- 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 [Lan77], 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:

- in state α, the station is idle, and its timer is set;
- in state β, the station is involved in an election and eligible;
- in state γ, the station is involved in an election but not eligible;
- in state α^*, the station is supposed to "generate a new token and switch immediately to state α".

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:

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

process ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, S:STATE) : noexit :=
   SUCC !CLAIM !Ai; (* timeout *)
      ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, BETA)
   []
   PRED !TOKEN;
      PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
   []
   PRED !CLAIM ?Aj:ADDR;
      (
```

```
14        [Ai < Aj] ->
15           SUCC !CLAIM !Aj;
16              ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S)
17        []
18        [Ai > Aj] ->
19           SUCC !CLAIM !Aj;
20              (
21              [S == BETA] ->
22                  ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, GAMMA)
23              []
24              [S <> BETA] ->
25                  ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S)
26              )
27        []
28        [Ai == Aj] ->
29           (
30           [S == BETA] ->
31              PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai)
32           []
33           [S <> BETA] ->
34              ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)
35           )
36        )
37 endproc
38
39 process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR) : noexit :=
40    OPEN !Ai;
41       CLOSE !Ai;
42          SUCC !TOKEN;
43             ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)
44    []
45    SUCC !TOKEN;
46       ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA)
47 endproc
```

In 1979, Ernest Chang and Rosemary Roberts proposed an improvement [CR79] 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[17]. 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[18]. 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[19]. Practically, Chang-Roberts' algorithm can be directly derived from Le Lann's algorithm by removing line 15 of the above LOTOS text.

[17] 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 report.

[18] In this report, 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 [CR79].

[19] 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 [Pet82, DKR82] with a worst case message bound of $O(n \log n)$.

We first tried to verify automatically a ring consisting of three Le Lann's stations connected with three reliable links[20]. We generated the corresponding graph using CÆSAR and compared it using ALDÉBARAN against the service graph (see Annex A under reference #3). ALDÉBARAN revealed that both graphs were not branching equivalent and produced a counter-example, a long execution sequence (178 transitions) accepted by the ring protocol. This sequence clearly violates mutual exclusion, since it ends with:

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

Using Chang-Roberts' stations instead, we performed the same experience (see Annex A under reference #4). The generated graph was smaller than previously, which is not surprising since Chang-Roberts' algorithm generates less messages than Le Lann's one. ALDÉBARAN reported that branching bisimulation was not satisfied either and, again, exhibited a long sequence (132 transitions) violating mutual exclusion:

$$...\tau^* \;.\; (\texttt{OPEN !A1}) \;.\; \tau^* \;.\; (\texttt{OPEN !A2})$$

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

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

where $i \neq j$ (see Annex A under references #16 and #17).

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*. [Lan77] 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. [Lan77] does not explain how to combine the precedence rule and the state machines. [CR79] 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.

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

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:

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

process ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, S:STATE, N:BOOL) : noexit :=
   [(S == ALPHA) and not (N)] ->
      SUCC !CLAIM !Ai; (* timeout *)
         ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, BETA, true)
   []
   PRED !TOKEN;
      PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, N)
   []
   PRED !CLAIM ?Aj:ADDR;
      (
      [Ai < Aj] ->
         SUCC !CLAIM !Aj;
            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S, N)
      []
      [Ai > Aj] ->
         SUCC !CLAIM !Aj;
            (
            [S == BETA] ->
                ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, GAMMA, N)
            []
            [S <> BETA] ->
                ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, S, N)
            )
      []
      [Ai == Aj] ->
         (
         [S == BETA] ->
            PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, false)
         []
         [S <> BETA] ->
            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, false)
         )
      )
endproc

process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, N:BOOL) : noexit :=
   OPEN !Ai;
      CLOSE !Ai;
         SUCC !TOKEN;
            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, N)
   []
   SUCC !TOKEN;
      ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, ALPHA, N)
endproc
```

Similarly, a modified version of Chang-Roberts' algorithm can be obtained by removing line 16 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 Annex A under references #5 and #6).

When replacing the reliable links with semi-reliable links (see Section 5.1), we also verified that branching equivalence was preserved (see Annex A under references #7 and #8).

However, when replacing the semi-reliable links with fully unreliable links (see Section 5.1), we observed that branching equivalence was not satisfied (see Annex 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 (see Annex A under references #18 and #19):

(`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 [Gau92].

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 privileged 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
5 process ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, C:BOOL, B:BOOL) : noexit :=
6    [C] ->
7       SUCC !CLAIM !Ai !B;
8          ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, B)
9    []
10   PRED !TOKEN;
11      PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, B)
12   []
13   PRED !CLAIM ?Aj:ADDR ?B0:BOOL;
14      (
15      [Ai < Aj] ->
16         SUCC !CLAIM !Aj !B0;
17            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, C, B)
18      []
19      [Ai > Aj] ->
20         SUCC !CLAIM !Aj !B0;
21            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, false, B)
22      []
23      [Ai == Aj] ->
24         (
25         [(B0 <> B) or not (C)] ->
26            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, C, B)
27         []
28         [(B0 == B) and C] ->
29            PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai, B)
30         )
31      )
32 endproc
33
34 process PRIVILEGE [OPEN, CLOSE, PRED, SUCC] (Ai:ADDR, B:BOOL): noexit :=
35   OPEN !Ai;
36      CLOSE !Ai;
37         SUCC !TOKEN;
38            ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, not (B))
39   []
40   SUCC !TOKEN;
41      ELECTION [OPEN, CLOSE, PRED, SUCC] (Ai, true, not (B))
42 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 16. 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 25 and 28 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 strong[21] equivalence) the graph of each station and each link separately. Then we used ALDÉBARAN BDD-based algorithms [FKM93] to prove that the ring was branching equivalent to the service graph (see Annex A under references #11 and #12).

We also studied variants of both algorithms in which line 6 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 Annex 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 Annex 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 / 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 [Lan77] and [CR79]: 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 [Lan77] this problem is solved by assuming the existence of a ring reconfiguration algorithm. In the token ring protocol [ISO95b], 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.

[21] 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 [PT87] implemented in ALDÉBARAN is more efficient.

This leads to a failure model much more severe than the one of [Lan77] and [CR79], 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 [CR79], but no solution is proposed. For the same purpose, [Lan77] 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:

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

where:

- the call to process "`ELECTION`" expresses the normal functioning mode of the station (this process will be detailed in Section 6.3);

- 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.

Formally, the "`FAIL`" process is defined as follows:

```
process FAIL [PRED, SUCC] (Ai:ADDR) : noexit :=
   PRED !TOKEN;
      SUCC !TOKEN;
         FAIL [PRED, SUCC] (Ai)
   []
   PRED !CLAIM ?Aj:ADDR ?B:BOOL [Ai <> Aj];
      SUCC !CLAIM !Aj !B;
         FAIL [PRED, SUCC] (Ai)
   []
   PRED !CLAIM !Ai ?B:BOOL;
      FAIL [PRED, SUCC] (Ai)
endproc
```

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:

  $$\tau^* \;.\; (\texttt{OPEN !}A_i) \;.\; \tau^* \;.\; (\texttt{CRASH !}A_i) \;.\; \tau^* \;.\; (\texttt{OPEN !}A_j)...$$

  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:

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

  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`".

```
SERVICE [OPEN, CLOSE, CRASH] ({} + A1 + A2 + A3)

where

process SERVICE [OPEN, CLOSE, CRASH] (E:ADDR_SET) : noexit :=
   choice Ai:ADDR []
      (
      [Ai isin E] ->
         (
         OPEN !Ai;
            SERVICE_BIS [OPEN, CLOSE, CRASH] (E, Ai)
         []
         CRASH !Ai;
            SERVICE [OPEN, CLOSE, CRASH] (E - Ai)
         )
      )
endproc

process SERVICE_BIS [OPEN, CLOSE, CRASH] (E:ADDR_SET, Ai:ADDR) : noexit :=
   CLOSE !Ai;
      SERVICE [OPEN, CLOSE, CRASH] (E)
   []
   CRASH !Ai;
      SERVICE [OPEN, CLOSE, CRASH] (E - Ai)
   []
   (
   choice Aj:ADDR []
```

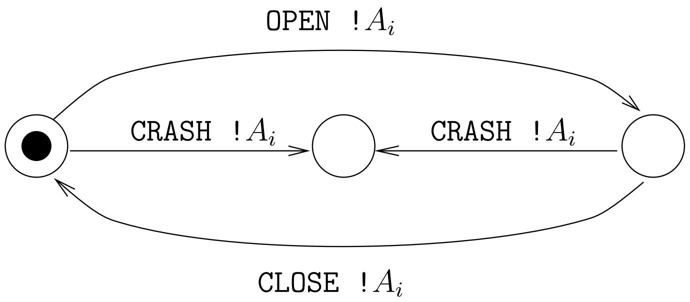

Figure 5: Graph expressing projection of service with crashes on station S_i

```
      (
      [(Aj isin E) and (Aj <> Ai)] ->
         CRASH !Aj;
            SERVICE_BIS [OPEN, CLOSE, CRASH] (E - Aj, Ai)
      )
   )
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 Figure 5.

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 6 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 6 and 16, by replacing lines 25 and 28 with "`[B0 <> B] ->`" and "`[B0 == B] ->`" respectively, and by removing all occurences of variable "`C`":

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

process ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai:ADDR, B:BOOL) : noexit  :=
   SUCC !CLAIM !Ai !B;
```

```
      ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, B)
   []
   PRED !TOKEN;
      PRIVILEGE [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, B)
   []
   PRED !CLAIM ?Aj:ADDR ?B0:BOOL;
      (
      [Ai < Aj] ->
         ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, B)
      []
      [Ai > Aj] ->
         SUCC !CLAIM !Aj !B0;
            ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, B)
      []
      [Ai == Aj] ->
         (
         [B0 <> B] ->
            ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, B)
         []
         [B0 == B] ->
            PRIVILEGE [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, B)
         )
      )
endproc

process PRIVILEGE [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai:ADDR, B:BOOL): noexit :=
   OPEN !Ai;
      CLOSE !Ai;
         SUCC !TOKEN;
            ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, not (B))
   []
   SUCC !TOKEN;
      ELECTION [OPEN, CLOSE, CRASH, PRED, SUCC] (Ai, not (B))
endproc

process FAIL [PRED, SUCC] (Ai:ADDR) : noexit :=
   PRED !TOKEN;
      SUCC !TOKEN;
         FAIL [PRED, SUCC] (Ai)
   []
   PRED !CLAIM ?Aj:ADDR ?B:BOOL [Ai <> Aj];
      SUCC !CLAIM !Aj !B;
         FAIL [PRED, SUCC] (Ai)
   []
   PRED !CLAIM !Ai ?B:BOOL;
      FAIL [PRED, SUCC] (Ai)
endproc
```

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 Annex 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 Figure 5, 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 Conclusion

To our knowledge, although the algorithms for distributed leader election last from the seventies and early eighties, they have been lacking a formal treatment until very recently. In this report, we have studied several variants of two existing leader election algorithms [Lan77, CR79] 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: [FGK95] gives a formal specification and proves the correctness of Dolev-Klawe-Rodesh's leader election algorithm for stations connected by a reliable, unidirectional ring; [BKKM95] and [BKKM96] 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 [CP88] 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 [Lan77, CR79] 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 report 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 [FGK95, BKKM95, BKKM96]. 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 report 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, Annex 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 Annex 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 [Lan77, FGK95, BKKM95, BKKM96], 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 that 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, and Radu Mateescu (INRIA/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) for his insightful comments about this report.

A Collected statistics

This Annex 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 following conventions are used. There are ten different kinds of stations, five different kinds of links, and four possible verification results:

abbreviation	*kind of station*	*see Section ...*
B	basic station	4.4
LL	Le Lann's original algorithm	5.2
CR	Chang-Roberts' original algorithm	5.2
LL_1	Le Lann's algorithm with first precedence rule	5.3
CR_1	Chang-Roberts' algorithm with first precedence rule	5.3
LL_2	Le Lann's algorithm with second precedence rule	5.4
CR_2	Chang-Roberts' algorithm with second precedence rule	5.4
LL_3	Same as LL_2 without the "`[C] ->`" guard	5.4
CR_3	Same as CR_2 without the "`C`" variable	5.4
F	Same as CR_3 with station crashes	6.3

abbreviation	*kind of link*	*see Section(s) ...*
T	Link carrying tokens reliably	4.3
$\underline{T}$	Link carrying tokens unreliably	5.1
$T.C$	Link carrying tokens and claims reliably	4.3
$\underline{T}.C$	Links carrying tokens unreliably and claims reliably	5.1 and 5.4
$\underline{T}.\underline{C}$	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

The following table 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).

reference	*station*	*link*	*states*	*trans*	*result*	*time*
#1	B	T	42	52	$\approx_B$	5
#2	B	$\underline{T}$	50	72	$\not\approx_B$ [d]	5
#3	LL	$T.C$	126,577	319,010	$\not\approx_B$ [e]	59
#4	CR	$T.C$	43,296	115,108	$\not\approx_B$ [e]	28
#5	LL_1	$T.C$	16,985	42,423	$\approx_B$	14
#6	CR_1	$T.C$	6,572	14,516	$\approx_B$	9
#7	LL_1	$\underline{T}.C$	30,085	77,680	$\approx_B$	24
#8	CR_1	$\underline{T}.C$	9,308	21,078	$\approx_B$	13

The following table 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 "$(s_1, t_1) \times (s_2, t_2) \times (s_3, t_3) \times (s_L, t_L)^3$" where s_i and t_i denote the number of states and transitions of the reduced graph for station S_i, and where 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).

ref.	*station*	*link*	*size of communicating processes*	*states*	*trans.*	*result*	*time*
#9	LL_1	$\underline{T}.\underline{C}$	$(15,27) \times (14,26) \times (13,25) \times (5,12)^3$	3,759	10,883	$\not\approx_B$ [d] $\approx_S$	48
#10	CR_1	$\underline{T}.\underline{C}$	$(9,21) \times (11,23) \times (13,25) \times (5,12)^3$	1,373	3,908	$\not\approx_B$ [d] $\approx_S$	29
#11	LL_2	$\underline{T}.\underline{C}$	$(16,32) \times (22,50) \times (18,46) \times (8,21)^3$	81,888	250,944	$\approx_B$	45
#12	CR_2	$\underline{T}.\underline{C}$	$(8,24) \times (14,42) \times (18,46) \times (8,21)^3$	9,568	32,808	$\approx_B$	31
#13	LL_3	$\underline{T}.\underline{C}$	$(16,32) \times (22,52) \times (18,48) \times (8,21)^3$	625,440	1,795,200	$\not\approx_B$ [e]	270
#14	CR_3	$\underline{T}.\underline{C}$	$(8,24) \times (12,28) \times (16,32) \times (8,21)^3$	10,512	33,896	$\approx_B$	38
#15	F	$\underline{T}.\underline{C}$	$(14,44) \times (18,52) \times (22,60) \times (18,96)^3$	162,995	609,297	$\approx_B$	69

The following table summarizes the measures obtained for on-the-fly verification using the OPEN/CÆSAR verification tools. For each experiment, the table gives: the kind of stations and links involved; the tool used; and the total time necessary for the complete verification process (in seconds).

reference	*station*	*link*	*on-the-fly tool*	*time*
#16	LL	$T.C$	EXHIBITOR	14
#17	CR	$T.C$	EXHIBITOR	13
#18	LL_1	$\underline{T}.\underline{C}$	TERMINATOR	14
#19	CR_1	$\underline{T}.\underline{C}$	TERMINATOR	11

References

[AL91] M. Abadi and L. Lamport. The Existence of Refinement Mappings. *Theoretical Computer Science*, 82(2):253–284, 1991.

[BB88] Tommaso Bolognesi and Ed Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14(1):25–59, January 1988.

[BFG+91] Ahmed Bouajjani, Jean-Claude Fernandez, Susanne Graf, Carlos Rodríguez, and Joseph Sifakis. Safety for Branching Time Semantics. In *Proceedings of 18th ICALP*. Springer Verlag, July 1991.

[BKKM95] 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, editors, *Proceedings of the Workshop on Algebra of Communicating Processes ACP'94*, Workshops in Computing, pages 338–357. Springer Verlag, 1995. Also available as Technical Report P9324, University of Amsterdam.

[BKKM96] J. J. Brunekreef, J.-P. Katoen, R. L. C. Koymans, and S. Mauw. Algebraic Specification of Dynamic Leader Election Protocols in Broadcast Networks. *Distributed Computing*, 9(4):157–171, January 1996.

[CGM+96] Ghassan Chehaibar, Hubert Garavel, Laurent Mounier, Nadia Tawbi, and Ferruccio Zulian. Specification and Verification of the PowerScale Bus Arbitration Protocol: An Industrial Experiment with LOTOS. In Reinhard Gotzhein and Jan Bredereke, editors, *Proceedings of the Joint International Conference on Formal Description Techniques* *for Distributed Systems and Communication Protocols, and Protocol Specification, Testing, and Verification FORTE/PSTV'96 (Kaiserslautern, Germany)*, pages 435–450. IFIP, Chapman & Hall, October 1996. Full version available as INRIA Research Report RR-2958.

[CP88] Ana Cavalli and Etienne Paul. Exhaustive Analysis and Simulation for Distributed Systems, Both Sides of the Same Coin. *Distributed Computing*, 2, 1988.

[CR79] Ernest Chang and Rosemary Roberts. An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes. *Communications of the ACM*, 22(5):281–283, may 1979.

[DKR82] D. Dolev, M. Klawe, and M. Rodeh. An $O(n \log n)$ Unidirectional Distributed Algorithm for Extrema Finding in a Circle. *Journal of Algorithms*, 3:245–260, 1982.

[dMRV92] Jan de Meer, Rudolf Roth, and Son Vuong. Introduction to Algebraic Specifications Based on the Language ACT ONE. *Computer Networks and ISDN Systems*, 23(5):363–392, 1992.

[EM85] H. Ehrig and B. Mahr. *Fundamentals of Algebraic Specification 1 — Equations and Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer Verlag, 1985.

[Fer90] Jean-Claude Fernandez. An Implementation of an Efficient Algorithm for Bisimulation Equivalence. *Science of Computer Programming*, 13(2–3):219–236, May 1990.

[FGK95] L. A. Fredlund, J. F. Groote, and H. P. Korver. Formal Verification of a Leader Election Protocol in Process Algebra. In *Proceedings of the Workshop on Algebra of Communicating Processes ACP'95*, pages 285–308, 1995. Computing Science Reports, Report 95-14, Eindhoven University of Technology. Also Technical Report R95-01, Swedish Institute of Computer Science.

[FGK+96] Jean-Claude Fernandez, Hubert Garavel, Alain Kerbrat, Radu Mateescu, Laurent Mounier, and Mihaela Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of the 8th Conference on Computer-Aided Verification (New Brunswick, New Jersey, USA)*, volume 1102 of *Lecture Notes in Computer Science*, pages 437–440. Springer Verlag, August 1996.

[FGM+92] Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodríguez, and Joseph Sifakis. A Toolbox for the Verification of LOTOS Programs. In Lori A. Clarke, editor, *Proceedings of the 14th International Conference on Software Engineering ICSE'14 (Melbourne, Australia)*, pages 246–259. ACM, May 1992.

[FKM93] Jean-Claude Fernandez, Alain Kerbrat, and Laurent Mounier. Symbolic Equivalence Checking. In C. Courcoubetis, editor, *Proceedings of the 5th Workshop on Computer-Aided Verification (Heraklion, Greece)*, volume 697 of *Lecture Notes in Computer Science*. Springer Verlag, June 1993.

[Gar89] Hubert Garavel. Compilation of LOTOS Abstract Data Types. In Son T. Vuong, editor, *Proceedings of the 2nd International Conference on Formal Description Techniques FORTE'89 (Vancouver B.C., Canada)*, pages 147–162. North-Holland, December 1989.

[Gar96] Hubert Garavel. An Overview of the Eucalyptus Toolbox. In Z. Brezočnik and T. Kapus, editors, *Proceedings of the COST 247 International Workshop on Applied Formal Methods in System Design (Maribor, Slovenia)*, pages 76–88. University of Maribor, Slovenia, June 1996.

[Gau92] Laurent Gauthier. Private communication. June 1992.

[GS90] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In L. Logrippo, R. L. Probert, and H. Ural, editors, *Proceedings of the 10th International Symposium on Protocol Specification, Testing and Verification (Ottawa, Canada)*, pages 379–394. IFIP, North-Holland, June 1990.

[GT93] Hubert Garavel and Philippe Turlier. CÆSAR.ADT : un compilateur pour les types abstraits algébriques du langage LOTOS. In Rachida Dssouli and Gregor v. Bochmann, editors, *Actes du Colloque Francophone pour l'Ingénierie des Protocoles CFIP'93 (Montréal, Canada)*, 1993.

[GV90] Jan Friso Groote and Frits Vaandrager. An Efficient Algorithm for Branching Bisimulation and Stuttering Equivalence. In M. S. Patterson, editor, *Proceedings of the 17th ICALP (Warwick)*, volume 443 of *Lecture Notes in Computer Science*, pages 626–638. Springer Verlag, 1990.

[Hoa85] C. A. R. Hoare. *Communicating Sequential Processes.* Prentice-Hall, 1985.

[ISO88a] ISO/IEC. File Transfer, Access and Management. International Standards 8571-*, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, 1988.

[ISO88b] 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, September 1988.

[ISO89a] ISO/IEC. LOTOS Description of the Session Protocol. Technical Report 9572, International Organization for Standardization — Open Systems Interconnection, Genève, 1989.

[ISO89b] ISO/IEC. LOTOS Description of the Session Service. Technical Report 9571, International Organization for Standardization — Open Systems Interconnection, Genève, 1989.

[ISO90] ISO/IEC. Local Area Networks — Part 4: Token-Passing Bus Access Method and Physical Layer Specifications. International Standard 8802-4, International Organization for Standardization — Information Processing Systems, Genève, 1990.

[ISO92a] ISO/IEC. Approved Algorithms for Message Authentication — Part 2: Message Authenticator Algorithm. International Standard 8731-2, International Organization for Standardization — Banking, Genève, 1992.

[ISO92b] ISO/IEC. Distributed Transaction Processing — Part 3: Protocol Specification. International Standard 10026-3, International Organization for Standardization — Information Technology — Open Systems Interconnection, Genève, 1992.

[ISO92c] ISO/IEC. Formal Description of ISO 8072 in LOTOS. Technical Report 10023, International Organization for Standardization — Telecommunications and Information Exchange between Systems, Genève, 1992.

[ISO92d] ISO/IEC. Formal Description of ISO 8073 (Classes 0, 1, 2, 3) in LOTOS. Technical Report 10024, International Organization for Standardization — Telecommunications and Information Exchange between Systems, Genève, 1992.

[ISO95a] ISO/IEC. Fibre Distributed Data Interface (FDDI). International Standards 9314-*, International Organization for Standardization — Information Processing Systems, Genève, 1989–1995.

[ISO95b] ISO/IEC. Local and Metropolitan Area Networks — Part 5: Token Ring Access Method and Physical Layer Specifications. International Standard 8802-5, International Organization for Standardization — Information Processing Systems, Genève, 1995.

[ISO95c] ISO/IEC. LOTOS Description of the CCR Protocol. Technical Report 11590, International Organization for Standardization — Open Systems Interconnection, Genève, 1995.

[ISO95d] ISO/IEC. LOTOS Description of the CCR Service. Technical Report 11589, International Organization for Standardization — Open Systems Interconnection, Genève, 1995.

[Lam77] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, 3(2):125–143, 1977.

[Lan77] Gérard Le Lann. Distributed Systems — Towards a Formal Approach. In B. Gilchrist, editor, *Information Processing 77*, pages 155–160. IFIP, North-Holland, 1977.

[LL95] R. Lai and A. Lo. An Analysis of the ISO FTAM Basic File Protocol Specified in LOTOS. *Australian Computer Journal*, 27(1):1–7, February 1995.

[Mil80] Robin Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[Mil89] Robin Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[Mou92] Laurent Mounier. *Méthodes de vérification de spécifications comportementales : étude et mise en œuvre*. Thèse de Doctorat, Université Joseph Fourier (Grenoble), January 1992.

[Mun91] Harold B. Munster. LOTOS Specification of the MAA Standard, with an Evaluation of LOTOS. NPL Report DITC 191/91, National Physical Laboratory, Teddington, Middlesex, UK, September 1991.

[Par81] David Park. Concurrency and Automata on Infinite Sequences. In Peter Deussen, editor, *Theoretical Computer Science*, volume 104 of *Lecture Notes in Computer Science*, pages 167–183. Springer Verlag, March 1981.

[Pet82] G. L. Peterson. An $O(n \log n)$ Unidirectional Algorithm for the Circular Extrema Problem. *ACM Transactions on Programming Languages and Systems*, 4(4):758–762, October 1982.

[PT87] Robert Paige and Robert E. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, December 1987.

[QS83] Jean-Pierre Queille and Joseph Sifakis. Fairness and Related Properties in Transition Systems — A Temporal Logic to Deal with Fairness. *Acta Informatica*, 19:195–220, 1983.

[Tan89] Andrew Tannenbaum. *Computer Networks*. Prentice-Hall, 1989.

[Tur93] Kenneth J. Turner, editor. *Using Formal Description Techniques – An Introduction to ESTELLE, LOTOS, and SDL*. John Wiley, 1993.

[vGW89] 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.

[Wes78] C.H. West. A General Technique for Communication Protocol Validation. *IBM Journal of Research and Development*, pages 393–404, July 1978.

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe - 38330 Montbonnot-St-Martin (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399