# Mutual Exclusion using a Token Ring

## 1 Introduction

The aim of this lab session is to illustrate LNT modeling and formal verification of a complex concurrent system using the tools available in the CADP toolbox[1]. In particular, we consider a distributed algorithm ensuring mutual exclusion on a token-ring network.

The directory `/matieres/5MMMVSC7/TD-CADP` contains several files with extensions `.lnt` and `.svl`, some of which have to be completed. Thus, before anything else, you should first make a local working copy of this directory. From now on, every file name refers to a file contained in your working copy.

To use CADP on the ENSIMAG computers[2] the `$CADP` and `$PATH` environment variables should be appropriately defined as described in the file `Cadp_env.sh`.

## 2 Expected Behavior

We consider a network consisting of $N$ *stations* competing for the access to a *shared resource*. The aim is to develop a distributed mechanism ensuring mutual exclusive access to the shared resource, i.e., at each instant at most one station is entitled to access the shared resource. A station being allowed to access the shared resource is said to be *priviledged*. For the sake of simplicity, we consider in the following a system with three stations ($N = 3$).

Each station $S_i$ has a unique address $A_i$. In LNT, the type of addresses Address and the related operations can be defined as follows (see file `DATA_TYPES.lnt`):

```
type Address is
   A1, A2, A3
   with ==, !=, >
end type
```

Considering only the accesses to the shared resource, i.e., abstracting all messages exchanged on the ring network, the behavior of a correctly functioning system will consist of a sequence of accesses to the shared resource. To study overlapping accesses, each access is split into two separate rendezvous on two gates with an offer of type Address: "OPEN ($A_i$)" represents the start of an access to the shared resource by station $A_i$, and "CLOSE ($A_i$)" represents the end of an access to the shared resource by station $A_i$. To ensure mutual exclusive accesses, each action "OPEN ($A_i$)" must be followed by an action "CLOSE ($A_i$)" (with the same $A_i$). Thus, these actions can be described by two gates OPEN and CLOSE of channel Resource, defined as follows (see file `DATA_TYPES.lnt`):

```
channel Resource is
  (Address)
end channel
```

All correct behaviors of the system are represented by the LNT model SERVICE (see file `SERVICE.lnt`):

[1]`http://cadp.inria.fr`

[2]CADP is available only in the room where the lab session will take place. The toolbox may be unavailable on some machines that were down during installation. Your teacher should have indicated the list of those machines. Do not hesitate to report any problem. If you want to work at home, you can ask for a free license for your own computer at `http://cadp.inria.fr/registration`. Please make sure to register using your student e-mail address (Grenoble INP or UGA).

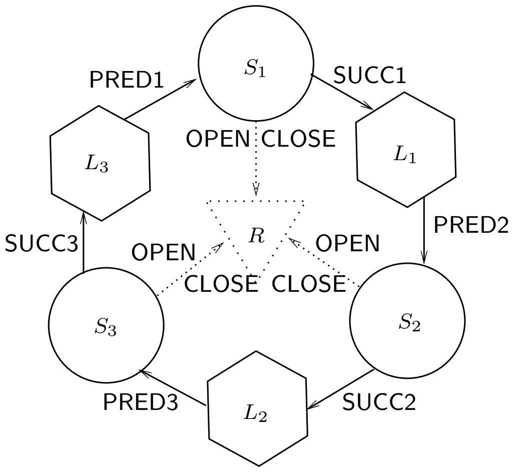

Figure 1: Ring network architecture

```
module SERVICE (DATA_TYPES) is

process MAIN [OPEN, CLOSE: Resource] is
  var ai: Address in
    loop
      OPEN (?ai); -- station ai accesses the resource
      CLOSE (ai)  -- station ai frees the resource
    end loop
  end var
end process

end module
```

Draw the LTS corresponding to the above LNT process MAIN.

## 3 Network Architecture

The ring network consists of three identical stations $S_1$, $S_2$, and $S_3$, connected by three identical unidirectional communication links $L_1$, $L_2$, and $L_3$. Consider the functions Succ and Pred, defined by the equations

$$\begin{aligned}\text{Succ}(i) &= (i \bmod 3) + 1\\ \text{Pred}(i) &= ((i+1) \bmod 3) + 1\end{aligned}$$

Then the link $L_i$ receives messages from station $S_i$ and forwards them to station $S_{\text{Succ}(i)}$. Thus, station $S_i$ can only receive messages from station $S_{\text{Pred}(i)}$. The network is illustrated by Figure 1.

The behavior of a station is described by an LNT process of the following profile:

```
process STATION [OPEN, CLOSE: Resource, PRED, SUCC: Ring] (a: Address, init: Bool)
```

where the gate and value parameters have the following meaning:

- The gates OPEN and CLOSE of channel Resource are used for the access to the shared resource, in the same way as for the previously seen process MAIN of module SERVICE.
- The gate PRED of channel Ring connects the station with its preceeding link, i.e., station $S_i$ is connected via PRED with link $L_{\text{Pred}(i)}$. Channel Ring is defined as follows, where Frame is a type that will be described later (see file `DATA_TYPES.lnt`):

```
channel Ring is
   (Frame)
end channel
```

- The gate SUCC of channel Ring connects the station with its succeeding link, i.e., station $S_i$ is connected via SUCC with link $L_i$.
- The parameter a of type Address defines the address of the station; thus, to model station $S_i$, the process STATION should be instantiated such that a is equal to $A_i$.
- The Boolean parameter init indicates whether the station is priviledged in the initial state. Obviously, to ensure exclusion, at most one station should be instantiated with its parameter init equal to true.[3]

The behavior of a communication link is described by an LNT process of the following profile:

```
process LINK [INPUT, OUTPUT: Ring]
```

where the gate parameters have the following meaning[4]:

- The gate INPUT of channel Ring connects the link with its preceding station, i.e., $L_i$ is connected by INPUT with station $S_i$.
- The gate OUTPUT of channel Ring connects the link with its succeding station, i.e., $L_i$ is connected by OUTPUT with station $S_{\mathrm{Succ}(i)}$.

The overall architecture is defined by process NETWORK in file NETWORK.lnt. Complete this process to instantiate a network with three stations and three links, in such a way that only the gates OPEN and CLOSE are visible ouside the process NETWORK. As a hint, this process should only contain process instantiations (of STATION and LINK)[5], parallel composition operators, and abstraction operators (**hide**).

## 4 Reliable Communication

To ensure mutual exclusive access to the shared resource, the token ring architecture uses a *token* circulating from station to station via the links. The station being currently in possession of the token is priviledged, and allowed to access the shared resource.

The transmission of the token from a station to a link, or vice-versa, is modeled by an action "$G$ (TOKEN)", i.e., as a rendezvous on a gate $G$ with (constant) offer TOKEN, one of the two constructors of type Frame, defined as follows (see file DATA_TYPES.lnt):

```
type Frame is
    TOKEN,
    CLAIM (a: Address, b: Bool)
end type
```

For now, the second constructor CLAIM() can be ignored.

In this section, we study reliable communication links. Thus, a link transmits each frame received on its INPUT gate eventually to its OUTPUT gate.

Under this assumption, the behavior of a station is simple. After receiving the token, a station is priviledged. A priviledged station can either retransmit the token towards the next station, or access the shared resource (using the gates OPEN and CLOSE as described before) and then retransmit the token. By transmitting the token, the station loses its priviledge, and has to wait for the token.

Obviously, both stations and links execute an infinite loop.

Complete process LINK in file LINK_1.lnt.

Complete process STATION in file BASIC_STATION.lnt.

[3]The choice which station should be priviledged is unspecified.

[4]It is not necessary to assign an address to the links

[5]Several different definitions of the body, i.e., the behavior, of the processes STATION and LINK will be provided and studied later.

Verify whether the LNT model PROTOCOL_1 contained in file PROTOCOL_1.lnt and consisting of the LNT modules DATA_TYPES, LINK_1, BASIC_STATION, and NETWORK is equivalent to the LTS generated from the LNT model SERVICE. This verification step can be done automatically using the command "`svl verify.svl`".

In particular, the following line in the SVL[6] script `verify.svl`:

```
check EQUIVALENCE (PROTOCOL_1, TRUE) ;
```

checks an SVL property called EQUIVALENCE, whose first parameter indicates the file containing the LNT model (here, PROTOCOL_1), and whose second parameter corresponds to the expected result (here, TRUE).

What would happen if no station were priviledged initially? Why? What would be the impact on the corresponding LTS? It is possible to generate the LTS in the BCG format[7] corresponding to an LNT model M using the command: "`lnt.open M generator M`", which stores the LTS in a file named `M.bcg`; the command "`bcg_edit M.bcg`"[8] allows the LTS to be visualized.

What would happen if several stations were priviledged initially?

Explore these questions by modifying your LNT specification and running the verification script.

## 5 Unreliable Communication

Consider now the case of unreliable communication links, which might arbitrarily lose messages instead of transmitting them to the next station. This loss is silent, so that neither sender nor receiver are aware or notified of the message loss.

An unreliable communication link behaves as follows.[9] In each iteration, after receiving a message, the link may either transmit the message or loose it by executing the silent action i instead.

Complete process LINK in file `LINK_2.lnt`. As before, the second constructor CLAIM() of type Frame can be ignored.

Add equivalence and model checking steps to the SVL script `verify.svl` to study the properties of the LNT model PROTOCOL_2 contained in file `PROTOCOL_2.lnt` and consisting of the LNT modules DATA_TYPES, LINK_2, BASIC_STATION, and NETWORK.

Study the differences of the verification results with those of PROTOCOL_1.

## 6 Fault-tolerant Algorithm

To provide a token ring for a network with unreliable communication links, G. Le Lann proposed an algorithm [Lan77, sections 3 et 4] which, after the loss of the token, enables a station entitled to regenerate a token to be elected. In 1979, E. Chang and R. Roberts proposed an improved version of Le Lann's algorithm, reducing the number of messages required for an election [CR79]. To obtain a network completely tolerant to message loss, we combine the algorithm of Chang&Roberts with the well-known alternating bit technique.[10]

In a nutshell, the algorithm proceeds as follows. Each station $S_i$ that did not receive the token for a certain amount of time may suppose that the token has been lost and emit a message with its address $A_i$ signaling that it requests the right to emit a new token. When a station receives its request to emit a token (after a complete round in the network), it is entitled to generate a new token. Each station filters requests to emit a token from stations with lower addresses. To avoid too numerous regenerations of tokens, each request to generate a new token is marked with a control bit enabling the station receiving the request after a complete round in the network to decide whether the request is obsolete.

The reception or emission of a request to reemit a token is modeled by a rendezvous of the form "$G$ (CLAIM ($A_i$, $B$))", where $G$ is a gate connecting a station to a communication link (or vice-versa), "CLAIM ($A_i$, $B$)" is a value of type Frame, $A_i$ is the address of the station emitting the request, and $B$ (of type Bool) is the control bit.

[6] `http://cadp.inria.fr/man/svl.html` and `http://cadp.inria.fr/man/svl-lang.html`
[7] `http://cadp.inria.fr/man/bcg.html`
[8] `http://cadp.inria.fr/man/bcg_edit.html`
[9] Among the different possibilities to model a faulty link, this corresponds to the most severe failure.
[10] Some other algorithms are discussed in [GM97].

As before, all stations have the same behaviour, differing only in their address ($A_1$, ..., $A_N$). In particular, all stations are entitled to request the emission of a new token.

There are two distinct states in the behaviour of a station: a priviledged state, written $\pi$, and an election state (indicating an ongoing election of a station to emit a token), written $\varepsilon$. Furthermore, each station has two local variables B and C of type Bool. Variable B is the control bit to stamp the request to emit a token.

Variable $C$ enables a station to avoid emitting a request if the station has already transmitted a request by a station with a higher address.

The behaviour of a station obeys the following rules:

(R0) Initially, a station is in the state $\varepsilon$, the value of B is undefined, and the value of C is true. Thus no station is priviledged, and no token is circulating in the network: the election algorithm will enable the generation of a token.

(R1) A station is in the priviledged state $\pi$ when it has received the token or when it is authorized to generate a token. In state $\pi$, a station may either retransmit the token to the next station, or first access the shared resource and then retransmit the token to the next station. The station goes into state $\varepsilon$, inverting the value of B and assigning the value true to C.

(R2) In state $\varepsilon$, a station changes into state $\pi$ upon reception of the token.

(R3) In state $\varepsilon$, a station changes into state $\pi$ upon reception of its own request with a control bit equal to B.

(R4) In state $\varepsilon$, a station ignores any reception of its own request with a control bit different from B.

(R5) In state $\varepsilon$, a station ignores any request by a station with a lower address.

(R6) In state $\varepsilon$, upon reception of a request by a station with a higher address, the station transmits this request to the next station and assigns the value false to C.

(R7) In state $\varepsilon$, a station can spontaneously emit a request if C is equal to true.

(R8) The state of a station does not change, except where explicitely stated in the previous rules. The value of B only changes by rule (R1). The value of C only changes by rules (R1) and (R6).

Notice that variable C is only meaningful in state $\varepsilon$.

Complete process LINK in file `LINK_3.lnt`, to model an unreliable communication link, which may loose tokens and requests: both kinds of frames can be lost silently.

Complete process STATION in file `ELECTION_STATION.lnt` to define a station according to the algorithm described above. Notice that this process has the same (gate and value) parameters as in file `BASIC_STATION.lnt`; this enables to keep all other LNT modules unchanged. Ensure that parameter init is no longer used in the body of process STATION.

Verify whether the LNT model PROTOCOL_3 contained in file `PROTOCOL_3.lnt` and consisting of the LNT modules DATA_TYPES, LINK_3, ELECTION_STATION, and NETWORK is equivalent to the LTS generated from the LNT model SERVICE.

## 7 Temporal Logic

Express the mutual exclusion property in MCL[11], and check it on the LNT models SERVICE, PROTOCOL_1, PROTOCOL_2, and PROTOCOL_3. This can be automated by uncommenting the MUTUAL_EXCLUSION property defined in the SVL script `verify.svl` and replacing the line containing "`...`" by the MCL propery (keeping the "`;`").

Could the verification results obtained in the previous section also be obtained using model checking of this mutual exclusion property? Are the verification results for the questions above the same?

[11] `http://cadp.inria.fr/man/mcl.html`

## References

[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.

[GM97] Hubert Garavel and Laurent Mounier. Specification and Verification of Various Distributed Leader Election Algorithms for Unidirectional Ring Networks. *Science of Computer Programming*, 29(1–2):171–197, July 1997. Special issue on Industrially Relevant Applications of Formal Analysis Techniques. Full version available as INRIA Research Report RR-2986.

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