Software Specification - Alloy Project

QS 2024/2025

In this project, we will use Alloy to model a simple broadcast protocol, referred to as OneAtATime. The goal of OneAtATime is to allow each of the participants of the protocol, referred to as *members*, to send messages to all other members with the restriction that only one member can send messages at a time. We refer to the member that can send messages as the *leader*.

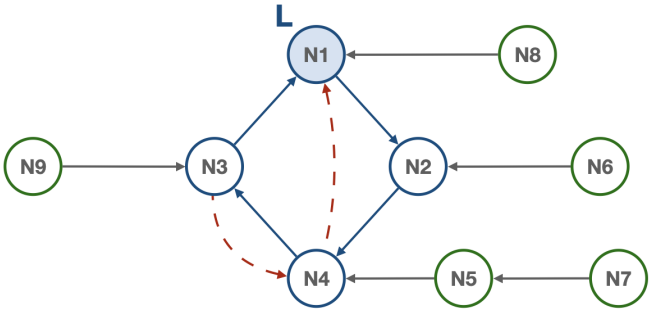
Protocol Topological Structure Besides the main message broadcasting operations, the protocol supports the following membership management operations:

- **Leader application:** A member that wants to broadcast messages can ask the leader to become the next leader. To this end, the leader keeps a queue of members that are waiting to become the next leader, referred to as *leader queue*. When a member asks to become the next leader, it is added to the end of the leader queue.
- **Leader promotion:** The leader can promote to leader the head of the *leader queue*. However, leader promotion can only happen if there is no ongoing message broadcast. When the head of the leader queue becomes the new leader, the the tail of the old leader's queue becomes the new leader's queue.

  EDIT: To simplify the modelling of the problem, assume that the leader only promotes another member to leader **after** having sent all its messages.
- **Member application:** Non-members can ask members to join in. To this end, each member keeps a queue of non-members that are waiting to join in, referred to as *member queue*.
- **Member promotion:** Any member can promote to member the head of its member queue.
- **Member exit:** With the exception of the leader, any member can leave the network. Upon leaving the network the member must transfer the elements in its member queue to another member's queue. In order to cease to be a member, the leader must first promote a member in the leader queue.

  EDIT: To simplify the modelling of the problem, you can further assume that a member may only cease to be a member if: **(1)** it is not in the leader queue; **(2)** its member queue is empty; and **(3)** all its messages are sent.
- **Non-member exit:** Non-members that are waiting to become members can drop out of their corresponding member queues without ever becoming a member.

Topological Constraints In order to facilitate message broadcast and membership management, the protocol ensures that node networks are organised in topologies such as the one illustrated below:



Where: **(1)** the leader is node N1; **(2)** the network members are nodes N1, N2, N3, and N4; **(3)** each network member has its own queue of nodes that have asked it to become a network member; for instance, the member queue of N4 consists of nodes N5 and N7; and **(4)** the leader queue is represented in dashed red arrows and consists of nodes N4 and N3.

Put formally, node networks must satisfy the following **topological constraints**: **(1)** members form a ring with each member pointing to another member (or itself); **(2)** each member queue consists of a (possibly empty) list of non-member nodes ending in the member in charge of that queue; **(3)** the leader queue consists of a list of member nodes ending in the leader; and **(4)** non-member nodes are not allowed to queue in more than one member queue at a time.

Message Broadcasting Having explained the network topology, we are now in a position to describe how message broadcasting works. Essentially, the leader sends a message to its next node and each node redirects the message to the following node until it reaches back the leader. When the leader receives the message back, the broadcasting of that message is considered closed. Hence, unlike network management operations, which can be thought of a single logical step, message broadcasting consists of three types of operations:

- **Broadcast initialisation:** The leader sends a message to its next node.
- **Message redirect:** A network member receives a message from its previous member and forwards it to the next member.
- **Broadcast termination:** The leader receives back one of the previously sent messages.

Accordingly, we say that messages can be in one of the following three states: **(i)** *pending* if their broadcasting process has not yet started; **(ii)** *sending* if their broadcasting process has started but not finished; **(iii)** *sent* if their broadcasting process has already finished. Importantly, a message can only be sent once, which means that, as the protocol progresses, the only allowed state transitions for messages are from *pending* to *sending* and from *sending* to *sent*.

Exercise 1: Static Modelling of OneAtATime Networks and Messages (6 val = 4 + 2) Consider the following partial static Alloy specification designed to describe the structure of *valid* OneAtATime networks:

```
sig Node {
  outbox: set Msg
}

sig Member in Node {
 nxt: one Node,
 qnxt : Node -> lone Node
}

one sig Leader in Member {
  lnxt: Member -> lone Member
}

sig LQueue in Member {}

sig Msg {
  sndr: Node,
  rcvrs: set Node
}

sig SentMsg, SendingMsg, PendingMsg in Msg {}
```

The specification consists of the following signatures:

- `Node` that represents network nodes; each node has an `outbox` field mapping the node to its own messages before being broadcasted and the messages that it must redirect;
- `Member` that represents member nodes; this signature has the fields `nxt` and `qnxt`,that respectively map each member to the next node in the ring and its member queue;
- `Leader` that represents the leader node; this signature has the field `lnxt` that associates the leader node with the leader queue;
- `LQueue` that represents the set of nodes contained in the leader queue;
- `Msg` that represents messages; this signature has the fields `sndr` and `rcvrs` that respectively associate each message with the node where it originates and the nodes to which it has already been delivered.
- `SentMsg`, `SendingMsg`, and `PendingMsg` model the three types of states that a message can be in.

Note that in order to be describe a *valid network* the provided signatures must not only satisfy the protocol's topological constraints but also ensure that messages are in a consistent state; for instance, a pending message cannot have been received by any node (because it was not yet sent). We refer to the constraints that guarantee that messages are in consistent states as *message-consistency constraints*.

1. Extend the provided static Alloy specification with the topological and message-consistency constraints required to ensure that it only describes valid networks. Each *fact* of your specification should be preceded by a comment describing the constraint that it is intended to model.
2. Use Alloy to generate two different network configurations. Both generated network configurations should have: **(1)** at least five nodes; **(2)** a non-empty leader queue; **(3)** at least two non-empty member queues; and **(4)** at least one message of each type of message state (i.e., one pending, one sending, and one sent). You should apply an Alloy theme to the generated model so as to guarantee that its presentation is consistent with the one used in the figure above. In particular, the only arrows to be displayed are those corresponding to: the main ring, the leader queue, and the member queues.

Hand-in Instructions: The solution of Exercise 1.1 should be given in a file named `Ex1.als`. Additionally, you should provide two images displaying the generated models for Exercise 1.2, respectively named `Ex1.2.1.png` and `Ex1.2.2.png`, together with the supporting Alloy theme file, `Ex1.thm`. Finally, you should clearly indicate in file `Ex1.als` the `run` commands used to generate both network configurations.

Exercise 2: Dynamic Modelling of OneAtATime Networks and Messages (9 val = 7 + 2)

1. Specify the dynamic system that describes the functioning of the OneAtATime protocol; your specification should include:
   - A predicate `init` describing the initial conditions of the protocol. Assume that in the beginning: **(1)** the set of members consists only of the leader; **(2)** all messages are in the pending state; and **(3)** no node is queueing to become a member.
   - State transformers for all network management operations: leader application, leader promotion, member application, member promotion, member exit, and non-member exit.
   - State transformers for all message routing operations: broadcast initialisation, message redirect, and broadcast termination.
   - Predicates `stutter`, `trans`, and `system` describing the whole behaviour of the system following the approach studied in the lectures.

2. Use Alloy to generate two system traces that illustrate the execution of the protocol. Both generated traces should have at least: **(1)** five nodes; **(2)** one leader promotion; **(3)** one member promotion; **(4)** one member exit; **(5)** one non-member exit; and **(6)** one complete message broadcast (i.e., a message that goes through the three message states: pending, sending, and sent).

Hand-in Instructions: The solution of Exercise 2.1 should be given in a file named `Ex2.als`. Additionally, you should provide two images displaying the generated traces for Exercise 2.2, respectively named `Ex2.2.1.png` and `Ex2.2.2.png`, together with the supporting Alloy theme file, `Ex2.thm`. Finally, you should clearly indicate in file `Ex2.als` the `run` commands used to generate both network traces.

Exercise 3: Verifying Properties of the OneAtATime Protocol (5 val)

1. Define a predicate `valid` that captures the topological and message-consistency constraints that valid networks must satisfy. Using this predicate, check that the dynamic system defined in Exercise 2 always maintains network validity.
   Hint: Essentially, it suffices to adapt the solution of Exercise 1.1 to the dynamic setting.

2. Define a predicate `fairness` that establishes the fairness conditions of the OneAtATime protocol. Essentially, every node should be able to join a member queue, become a member, become a leader, and send its messages.

3. Using the fairness predicate, model and check the following liveness properties of OneAtATime:

   (a) **EDIT:** In a network with at least two nodes, under fairness conditions and if no exit operations are performed (both member exit and non-member exit), then all message broadcasts terminate.

   (b) **EDIT:** ~~Under fairness conditions, if no node leaves the network after a message is sent, then all the nodes in the network at the time of the broadcast initialisation will receive the message.~~

4. **EDIT:** Consider the following variation of property 3.a:

   > In a network with at least two nodes, under fairness conditions, all message broadcasts terminate.

   Use Alloy to generate a trace that disproves this property.

Hand-in Instructions: The Alloy solutions of Exercises 3.1-3.4 should be given in a file named `Ex3.als`. Within this file, please include the `check` commands used in Exercises 3.3 and 3.4. Additionally, you should provide one image displaying the generated trace for Exercise 3.4, named `Ex3.png`.

Instructions

Hand-in Instructions The project is due on the 25th of October, 2024. Be sure to create a **zip** file containing all the answer files and upload it in Fenix. Submissions will be closed at 23h59 on the 25th of October, 2024. Do not wait until the last few minutes for submitting the project.

Project Discussion After submission, you may be asked to present your work so as to streamline the assessment of the project as well as to detect potential fraud situations. During this discussion, you may be required to perform small changes to the submitted code.

Fraud Detection and Plagiarism The submission of the project assumes the commitment of honour that the project was solely executed by the members of the group that are referenced in the files/documents submitted for evaluation. Failure to stand up to this commitment, i.e., the appropriation of work done by other groups or someone else, either voluntarily or involuntarily, will have as consequence the immediate failure of this year's Software Specification course for all students involved (including those who facilitated the occurrence).