# SSFT '22: UCLID5 Lab

Federico Mora and Sanjit A. Seshia

University of California, Berkeley

The goal of this lab is to model and verify a distributed protocol called the ring leader election protocol. The ring leader election protocol consists of a set of nodes with unique labels communicating in a ring: each node can only receive messages from the node on its "left" and can only send messages to the node on its "right." The goal of the protocol is for the nodes to collectively discover the node with the greatest label. The following pseudo-code describes the behaviour of individual nodes.

1. Each node begins by sending its own label value to its right.
2. From then on, nodes react to messages that they receive.
   (a) If a node receives a label value greater than its own label value, then it forwards that received label value to its right.
   (b) If the received label value is smaller than its own label value, then the node sends its own label value to the right.
   (c) Finally, if a node ever receives its own label value, then it declares itself the winner (by setting a local, boolean flag).

*Task 1: Model A Ring Leader Election Node.* Fill in the incomplete UCLID5 module in Fig. 1 following the description of ring leader election nodes above.

```
module node {
    // TODO: declare local variables.
    // Hint: use "input", "output", and "var" keywords.
    // Hint: use a boolean flag to indicate that the node "won."

    init {
        // TODO: define node initialization.
        // Hint: don't worry about label uniqueness yet.
    }

    next {
        // TODO: define transition relation.
    }
}
```

**Fig. 1.** Template for Task 1.

*Task 2: Model The Asynchronous Composition Using Interleaving Semantics of Three Nodes.* Fill in the incomplete UCLID5 module in Fig. 2 following the description of how ring leader election nodes interact and execute above.

```
module main {
    // TODO: type declarations.
    // Hint: declare an enum for "taking turns" later.

    // TODO: declare local variables.
    // Hint: you'll need local variables to interact with instances.

    // TODO: declare instances of other modules.
    // Hint: use the "instance" keyword.
    // Hint: the input of one instance can be the output of another.

    init {
        // TODO: define system initialization.
        // Hint: now is a good time to worry about label uniqueness.
        // Hint: use the "assume" keyword.
    }

    next {
        // TODO: define transition relation.
        // Hint: use the "havoc" keyword.
        // Hint: use the "case" keyword.
    }
}
```

**Fig. 2.** Template for Task 2.

*Task 3: Specify Your Ring Leader Election Model.* Use the "invariant" keyword to specify that there is only ever a single winner and that the winner holds the greatest label value. What is the simplest way to specify these properties? Does it matter which of the three nodes is the winner? What kind of properties are these? Safety? Liveness? Does your model account for messages being "dropped" during execution (e.g., in real life a text message might not make it to its destination)? Does your model account for messages being "tampered with" during execution (e.g., in real life a man-in-the-middle attack might tamper with a text message)? Does your model account for messages being "duplicated" during execution (e.g., in real life a text message might be sent twice)? Do any of these modelling considerations matter for your specification?

*Task 4: Use Bounded Model Checking To Sanity Check Your Model And Spec.* First, use bounded model checking to ensure that your model satisfies your specification for a few steps (you will need to add a control block and use the "bmc" command). Second, use bounded model checking and the "assert" keyword to ensure that it is possible for a node to win (what argument should you give to "assert" and where should you put the "assert" statement?). If a node winning is not reachable, you may need to revise your model. Third, use bounded model checking and the "assert" keyword to find the minimum number of steps required for a leader to be elected. If we had $n$ nodes, what would the minimum number of required steps to find a leader be?

*Task 5: Use Induction To Ensure Your Model Satisfies Your Specifications.* Change the "bmc" command in the control block to "induction" and run UCLID5. You should get a counterexample to induction. Add auxiliary specifications using the "invariant" keyword to block this counterexample. Repeat this process until you make the proof by induction pass (until the conjunction of properties is inductive). Beware! This is hard stuff! Try to think of general, high-level properties that will make the proof pass.