Formal Verification of Distributed Algorithms using Distributed PlusCal

Student: Heba Al-Kayed
Supervisors : Stephan Merz, Horatiu Cirstea

Loria
Laboratoire lorrain de recherche en informatique et ses applications

DEPEND
Erasmus Mundus Joint Masters Degree
MSc in Advanced Systems Dependability
Maynooth University National University of Ireland Maynooth
University of St Andrews
UNIVERSITÉ DE LORRAINE

Motivations

➢ **Formal Methods for Distributed Algorithms**

- Distributed Algorithms are prone to deadlocks and race conditions.
- Formal verification methods have been employed successfully to model the system and its properties and then verify its correctness.

➢ **Modeling Distributed Algorithms**

- TLA+[1] is a formal language used to describe algorithms.
- TLA+ relies on mathematical logic, it is used to specify and verify complicated algorithms concisely.
- PlusCal[2] was designed as an algorithm language with a more familiar syntax that can be translated into TLA+ specifications.
- PlusCal[2] lacks primitives for communication between processes which makes modeling distributed algorithms difficult.

References

[1]L.Lamport, (2000). Specifying Concurrent Systems with TLA+.
[2]L. Lamport, (2009). The PlusCal Algorithm Language, ICTAC 2009

Contributions

➢ We introduce Distributed PlusCal, an extension of PlusCal for modeling distributed algorithms.

➢ Distributed PlusCal offers primitives that aid in modeling distributed algorithms such as

- Sub-processes in the form

```
process(…) { \* sub-process
    …
}{ \* sub-process
  …
}
```

- Communication channels :
  - Unordered channels with the syntax
    channel *⟨identifier⟩[⟨dimension1,…dimensionN⟩]*
  - FIFO channels with the syntax
    fifo *⟨identifier⟩[⟨dimension1,…dimensionN⟩]*
  - Channels support the following operations
    1. send
    2. broadcast
    3. multicast
    4. receive
    5. clear

➢ We provide a backward compatible translator that translates from Distributed PlusCal and PlusCal to TLA+.

```
(***
--algorithm TwoPhaseCommit {

channels agt[Agent], coord;

  fair process (a \in Agent)
  variable aState = "unknown"; { \* main sub-process

    a1: if (aState = "unknown") {
        with(st \in {"accept", "refuse"})
          aState := st;
          send(coord, [type |-> st, agent |-> self]);};};
    a2: await(aState \in {"commit", "abort"})
  } { \* message handling sub-process
    a3:await (aState # "unknown");
       receive(agt[self], aState);
  }

  fair process (c = Coord)
  variables cState = "unknown",
            commits = {},
            msg = {}; {
   c1: await(cState \in {"commit", "abort"});
       broadcast(agt, [ag \in Agent|-> cState]);
  } {
     c2:while (cState \notin {"abort", "commit"}) {
        receive(coord, msg);
            if (msg.type = "refuse") {
                cState := "abort";
            }
            else if (msg.type = "accept") {
                commits := commits \cup {msg.agent};
                 if (commits = Agent) {
                    cState := "commit";
                }
             }
       }
    }
}***)
```