

Formal Verification of Distributed Algorithms using Distributed PlusCal

Heba Al-kayed

Supervisors: Stephan Merz, Horatui Cirstea

Université de Lorraine

6th July 2020

Introduction

Formal methods for distributed algorithms

- ▶ Distributed systems are based on continuous interactions among components.
- ▶ They produce bugs that are difficult to find by testing
- ▶ Formal verification techniques like model checking have been employed successfully to model the system and its properties and then verify its correctness

TLA⁺

- ▶ A formal specification language in which algorithms and systems can be described at a high level of abstraction and can be formally verified
- ▶ TLA⁺ specifications usually have the form

$$Init \wedge \Box [Next]_{vars} \wedge L$$

- ▶ *Init* and *Next* are predicates, and *L* is a liveness or fairness property.

TLA⁺ ToolBox

The TLA⁺ Toolbox is an IDE for the TLA⁺ tools

- ▶ Create and edit TLA⁺ specification
- ▶ Run the PlusCal translator
- ▶ Run the TLC model checker
- ▶ Run the TLA⁺ proof system

PlusCal

- ▶ Designed as an algorithm language with a more familiar syntax
- ▶ Can be translated into TLA+ specifications
- ▶ PlusCal can describe both concurrent and sequential algorithms

```
(*  
--algorithm SemaphoreMutex {  
variables sem = 1;  
  process(p \in 1..N)  
  {  
    start : while (TRUE){  
      enter :   when (sem > 0);  
               sem := sem - 1;  
      cs :     skip ;  
      exit :   sem := sem + 1 ;  
    }  
  }  
}  
*)
```

PlusCal to TLA⁺ translator

► Generating the TLA⁺ Init predicate

```
Init == (* Global variables *)
        /\ sem = 1
        /\ pc = [self \in ProcSet |->
                  "start"]
```

► Translating each PlusCal label into a TLA⁺ action

```
st(self) == /\ pc[self] = "st"
             /\ pc' = [pc EXCEPT
                       ![self] = "enter"]
             /\ sem' = sem
```

```
(*
--algorithm SemaphoreMutex {
variables sem = 1;
process(p \in 1..N)
{
  start : while (TRUE){
  enter :   when (sem > 0);
           sem := sem - 1;
  cs :     skip ;
  exit :   sem := sem + 1 ;
        }
}
*)
```

PlusCal to TLA⁺ translator

- Generate the TLA⁺ Next predicate

```
Next == (\E self \in 1..N: st(self)
  \/\ enter(self)
  \/\ cs(self)
  \/\ exit(self))
```

- Generate the complete specification Spec

```
Spec == Init /\ [][Next]_vars
```

```
(*
--algorithm SemaphoreMutex {
variables sem = 1;
process(p \in 1..N)
{
  start : while (TRUE){
  enter :   when (sem > 0);
           sem := sem - 1;

  cs :     skip ;
  exit :   sem := sem + 1 ;
           }
  }
}
*)
```

Distributed PlusCal

- ▶ An extension of PlusCal offering constructs for modeling distributed algorithms
 - ▶ Sub-processes
 - ▶ Communication channels
- ▶ A compiler to translate to TLA⁺

General Structure of an algorithm

```
(* algorithm <algorithm name>

(* Declaration section *)
variables <variable declarations>
channels <channel declarations>
fifos <fifo declarations>

(* Definition section *)
define <definition name> == <definition description>

(* Macro section *)
macro <name>(var1, ...)
  <macro-body of statements>

(* Procedure section *)
procedure <name>(arg1, ...)
  variables <local variable declarations>
  <procedure body of statements>

(* Processes section *)
process (<name> [=|\in] <Expr>))
  variables <variable declarations>
  <sub-processes>
*)
```

Communication Channels

- ▶ classified by the way they handle the addition and removal of messages
 - ▶ Unordered channels
 - ▶ FIFO channels
- ▶ Supported operators
 - ▶ send
 - ▶ receive
 - ▶ broadcast
 - ▶ multicast
 - ▶ clear

Sub-Processes

- ▶ Enables the process to execute multiple tasks in parallel.
- ▶ Each sub-process consists of labeled statements.
- ▶ The special variable *pc* was modified to have the following definition

$$pc = [self \in ProcSet \mapsto [self \in IdSet \mapsto \langle "lbl", \dots \rangle]]$$

where *ProcSet* is a set that contains all the process identifiers, and the labels that appear in the sequence are the entry point actions for each sub-process.

Distributed PlusCal Example

```
1      /* message channels
2      channels coord, agt[Agent];
3
4      fair process (a \in Agent)
5      variable aState = "unknown"; { /* main sub-process
6
7      a1: if (aState = "unknown") {
8          with(st \in {"accept", "refuse"}) {
9              aState := st;
10             send(coord, [type |-> st, agent |-> self]);
11         };
12     };
13     a2: await(aState \in {"commit", "abort"})
14
15 } { /* message handling sub-process
16
17     a3: await (aState # "unknown");
18     receive(agt[self], aState);
19 }
```

Translation to TLA+

```
1  a3:await (aState # "unknown");  
2  receive(agt[self], aState);
```

```
1  a3(self) == /\ pc[self] [2] = "a3"  
2              /\ (aState[self] # "unknown")  
3              /\ \E ag \in agt[self]:  
4                  /\ aState' = [aState EXCEPT ![self] = ag]  
5                  /\ agt' = [agt EXCEPT ![self] = agt[self] \ {ag}]  
6              /\ pc' = [pc EXCEPT ![self] = [0 EXCEPT ![2] = "a4"]]  
7              /\ UNCHANGED << coord, cState, commits, msg >>
```

Contributions and future work

Contributions

- ▶ An extension of PlusCal called Distributed PlusCal
- ▶ Distributed PlusCal offers constructs that are designed for modeling distributed algorithms
- ▶ A backward compatible translator that translates from Distributed PlusCal and PlusCal to TLA+

Future Work

In the future we aim to introduce more types of communication channels and channel operators, and we're working on supporting PlusCals p-syntax as well.