

An Extension of PlusCal for Modeling Distributed Algorithms

Heba Alkayed, Horatiu Cirstea, Stephan Merz

University of Lorraine, CNRS, Inria, Nancy, France

September 18, 2020

Formal Specification Languages

- ▶ Algorithms modeled using TLA⁺ can be formally verified using the TLA⁺ Toolbox
- ▶ PlusCal algorithms have a more familiar syntax and can be translated to TLA⁺

Distributed PlusCal Algorithms

Motivation

An extension of PlusCal with a syntax that offers constructs for modeling distributed algorithms naturally

Features

- ▶ Introduces
 - ▶ Sub-processes
 - ▶ Communication channels
- ▶ Can be translated into a TLA+ specification

Motivating example

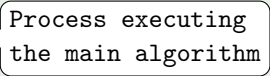
Lamport Mutex Algorithm

- ▶ An algorithm for Mutual Exclusion in Distributed Systems
- ▶ Critical section requests are ordered based on timestamps
- ▶ Processes exchange 3 types of messages
 - ▶ Request
 - ▶ Acknowledge
 - ▶ Release
- ▶ Processes need to asynchronously receive messages from each other

Process modelisation (in PlusCal)

Lamport Mutex Example - PlusCal

```
(**--algorithm LamportMutex {  
  process (proc \in Proc) {  
    ncs: while (TRUE) {  
      /* non-critical section */  
    try: /* multicast a message requesting access to cs */  
    enter: /* wait for acknowledgements */  
    cs: /* critical section */  
    exit: /* multicast the release message */  
  } /* end while */  
} /* end process */
```



Process executing
the main algorithm

Process modelisation (in Distributed PlusCal)

Lamport Mutex Example - PlusCal

```
process (comm \in Comm) {
```

Process handling
messages

```
  rcv: while (TRUE) {
```

```
    with (prc = node(self),
```

```
      ...) {
```

```
        /* handle request, acknowledge and release  
        messages
```

```
      }
```

```
    } /* end while
```

```
  } /* end process
```

Process modelisation (in Distributed PlusCal)

Lamport Mutex Example - PlusCal

```
process (comm \in Comm) {  
  rcv: while (TRUE) {  
    with (prc = node(self),  
        ...) {  
      /* handle request, acknowledge and release  
      messages  
    }  
  } /* end while  
} /* end process  
**)
```

Proc == 1 .. N
Comm == N+1..N+N
node(c) == c - N

Lamport Mutex in Distributed PlusCal

Lamport Mutex Example - Distributed PlusCal

```
fifos network[Proc, Proc];
process(p \in Proc)
  variables ..
{
  ncs: /*non-critical section
  ..
  exit: /* multicast the
        /* release message
} {

  rcv: /* receive msg from channel
        /* handle request, acknowledge and release
        messages
} /* end message handling thread
**)
```

sub-process executing
the main algorithm

message handling
sub-process

Channels modelisation

Declaration (in PlusCal)

```
network=[p,q \in Proc |->  $\langle \rangle$ ]
```

Channels modelisation

Declaration (in PlusCal)

```
network=[p,q \in Proc |-> <>]
```

Declaration (in Distrbuted PlusCal)

```
fifos network[Proc, Proc];
```

Channels modelisation

Declaration (in PlusCal)

```
network=[p,q \in Proc |-> <>]
```

Declaration (in Distrbuted PlusCal)

```
fifos network[Proc, Proc];
```

Operation (in PlusCal)

```
macro mcast(p, msg) {  
  network := [s,d \in Proc  
    |-> IF s = p /\ d # p  
    THEN  
      Append(network[s,d],  
        msg) ELSE network[s,d]]  
}  
mcast(self, Request(clock));l
```

Channels modelisation

Declaration (in PlusCal)

```
network=[p,q \in Proc |-> <>]
```

Declaration (in Distrbuted PlusCal)

```
fifos network[Proc, Proc];
```

Operation (in PlusCal)

```
macro mcast(p, msg) {  
  network := [s,d \in Proc  
    |-> IF s = p /\ d # p  
    THEN  
      Append(network[s,d],  
        msg) ELSE network[s,d]]  
}  
mcast(self, Request(clock));1
```

Operation (in Distrbuted PlusCal)

```
/* the 1st argument is the  
   channel name and the 2nd is  
   a TLA+ expression that  
   specifies the message and  
   the intended recipients  
  
multicast(network, [self, p \in  
  Proc |-> Request(clock)]);
```

General Structure of an algorithm

```
(* --algorithm <algorithm name>
(* Declaration section *)
variables <variable declarations>
channels <channel declarations>
fifos <fifo declarations>
(* ... *)
(* Processes section *)
process (<name> [=|\in] <Expr>))
    variables <variable declarations>
    <subprocesses>
*)
```

Operation on channels

- ▶ Supported operators
 - ▶ `send(ch, el)`
 - ▶ `receive(ch, var)`
 - ▶ `broadcast(ch, [x \in S \mapsto e(x)])`
 - ▶ `multicast(ch, [x \in S \mapsto e(x)])`
 - ▶ `clear(ch)`

Unordered Channels Translation

channel $\langle id \rangle [\langle Expr_1 \rangle, \dots, \langle Expr_N \rangle];$

► Translation based on TLA⁺ sets

► $\text{send}(\text{chan}[e], \text{msg}) \triangleq$
 $\text{chan}' = [\text{chan} \text{ EXCEPT } ![e] = \text{chan}[e] \setminus \text{cup } \{\text{msg}\}]$

► $\text{receive}(\text{chan}[e], \text{var}) \triangleq \backslash E \text{ temp } \backslash \text{in } \text{chan}[e]:$
 $\quad \backslash \text{var}' = \text{temp}$
 $\quad \backslash \text{chan}' = [\text{chan} \text{ EXCEPT } ![e]$
 $\quad \quad \quad = \text{chan}[e] \setminus \{\text{temp}\}]$

FIFO Channels Translation

fifo $\langle id \rangle [\langle Expr_1 \rangle, \dots, \langle Expr_N \rangle];$

► Translation based on TLA⁺ sequences

- $\text{send}(\text{chan}[e], \text{msg}) \triangleq$
 $\text{chan}' = [\text{chan} \text{ EXCEPT } ![e] = \text{Append}(@, \text{msg})]$
- $\text{receive}(\text{chan}[e], \text{var}) \triangleq$ $\wedge \text{Len}(\text{chan}[e]) > 0$
 $\wedge \text{var}' = [\text{Head}(\text{chan}[e])]$
 $\wedge \text{chan}' = [\text{chan} \text{ EXCEPT } ![e]$
 $= \text{Tail}(@)]$

Program counter

- ▶ The special variable pc was modified to take into account sub-processes

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

where $IdSet$ is a collection that contains process identifiers, and the labels that appear in the sequence are the entry point actions for each sub-process

Translation to TLA⁺

```
exit:
  clock := clock + 1;
  multicast(network, [self, p \in Proc \ {self} |->
    Release(clock)]);
```

```
exit(self) ==
  /\ pc[self][1] = "exit"
  /\ clock' = [clock EXCEPT ![self] = clock[self] + 1]
  /\ network' = [<<slf, n>> \in DOMAIN network |->
    IF
      slf = self /\ p \in Proc \ { self }
    THEN
      Append(network[slf, p], Release(clock'[self]))
    ELSE
      network[slf, p]]
  /\ pc' = [pc EXCEPT ![self][1] = "ncs"]
  /\ UNCHANGED << req, ack, sndr, msg >>
```

Contributions and future work

Contributions

- ▶ An extension of PlusCal offering the possibility to define
 - ▶ Sub-Processes
 - ▶ Communication Channels
- ▶ A backward compatible translator to TLA⁺

Future Work

In the future we aim to introduce more types of communication channels and channel operators.