An Extension of PlusCal for Modeling Distributed Algorithms

Heba Alkayed, Horatiu Cirstea, Stephan Merz

University of Lorraine, CNRS, Inria, Nancy, France

September 17, 2020

Introduction

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

Comparison through Lamport Mutex Algorithm

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

Lamport Mutex in PlusCal

Lamport Mutex Example - PlusCal

```
(**--algorithm LamportMutex {
variables
(* FIFO message passing between pairs of nodes *)
network = [p,q \in Proc |-> << >>],

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
```

Lamport Mutex in 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
```

Lamport Mutex in PlusCal

Lamport Mutex Example - PlusCal

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

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

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: \* handle request, acknowledge and release
    messages
} \* end message handling thread
**)
```

sub-process executing
the main algorithm

message handling
sub-process

Comparison in terms of communication channels

PlusCal

```
network = [p,q \in Proc |-> ⟨⟩]
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));
```

Distributed PlusCal

```
fifos network[Proc, Proc];
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>
*)
```

Communication Channel Translation

- Classified by the way they handle the addition and removal of messages
    - Unordered channels
    - FIFO 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** or **channels**, as shown below.

$$\textbf{channel}\ \langle \mathit{id} \rangle[\langle \mathit{Expr}_1 \rangle, \ldots, \langle \mathit{Expr}_N \rangle];$$

- based on TLA$^+$ sets
- Operator translation to TLA$^+$
  - `send(chan[e], msg)` $\triangleq$

    ```
    chan' = [chan EXCEPT ![e] = chan[e] \cup {msg}]
    ```

  - `receive(chan[e], var)` $\triangleq$

    ```
    \E temp \in chan[e]:
    /\ var' = temp
    /\ chan' = [chan EXCEPT ![e]
                = chan[e] \ {temp}]
    ```

FIFO Channels Translation

- **fifo** or **fifos**, as shown below.

$$\textbf{fifo}\ \langle id\rangle[\langle Expr_1\rangle, \ldots, \langle Expr_N\rangle];$$

- based on TLA$^+$ sequences
- Operator translation to TLA$^+$
  - `send(chan[e], msg)` $\triangleq$
    ```
    chan' = [chan EXCEPT ![e] = Append(@, msg)]
    ```
  - `receive(chan[e], var)` $\triangleq$
    ```
    /\ Len(chan[e]) > 0
    /\ var' = [Head(chan[e])]
    /\ chan' = [chan EXCEPT ![e]
                    = Tail(@) ]
    ```

Sub-Processes Translation

- The special variable *pc* was modified to have be initialized as

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

  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 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.