An Extension of PlusCal for Modeling Distributed Algorithms

Heba Alkayed, Horatiu Cirstea, Stephan Merz

University of Lorraine, CNRS, Inria, Nancy, France

September 19, 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

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 model (in PlusCal)

Lamport Mutex Example - PlusCal

```
\* Variables must be declared globally to be used by the
    inter-playing processes representing this algorithm
variables network, clock ...

(**--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 model (in PlusCal)

Lamport Mutex Example - PlusCal

Process handling messages

```
process (comm \in Comm) {

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

Process model (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
**)
```

```
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

Modeling channels

Declaration (in PlusCal)

```
network=[p,q \in Proc |-> ⟨⟩]
```

Modeling channels

Declaration (in PlusCal)

```
network=[p,q \in Proc |-> ⟨⟩]
```

Declaration (in Distrbuted PlusCal)

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

Modeling channels

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));
```

Modeling channels

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));
```

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>
*)
```

Operations 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)`

Translation of Unordered Channels

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

- Translation based on TLA$^+$ sets
  - `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}]
    ```

Translation of FIFO Channels

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

- Translation based on TLA$^+$ sequences
  - `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(@)]`

Program counter

- The variable *pc* is indexed by processes and sub-processes

```
pc = [self \in ProcSet|->
                        IF self \in P_1
                        THEN << lbl_{11}, lbl_{12}, ...>>
                        IF IF ...]
```

where the lbl_{ij} are the entry labels of the subprocesses of the process type P_i.

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, p>> \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.