An Extension of PlusCal for Modeling Distributed Algorithms

Heba Alkayed, Horatiu Cirstea, Stephan Merz

University of Lorraine, CNRS, Inria, Nancy, France

September 28, 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’s Mutex Algorithm

- An algorithm for Mutual Exclusion in Distributed Systems
- Critical section requests are ordered based on logical clocks
- Processes exchange 3 types of messages
  - Request
  - Acknowledge
  - Release
- Processes asynchronously receive messages from each other
Algorithm in PlusCal: main process

\* 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
Algorithm in PlusCal: helper process

```
process (comm \in Comm) {

  rcv: while (TRUE) {
    with (prc = node(self), ...) {
      /* handle request, ack and release messages */
    }
  } /* end while */
} /* end process */
```
Algorithm in PlusCal: helper process

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

Proc == 1 .. N
Comm == N+1..N+N
node(c) == c - N
fifos network[Proc, Proc];
process(p \in Proc)  
  variables ..
{
  ncs: while (TRUE) {
  \*non-critical section
  ...
  exit: \* multicast the
  \* release message
  } \* end while
} \* end message handling thread
} \* end message handling thread
**)
Modeling channels

Declaration (in PlusCal)

\[
\text{network=} [p,q \ \in \text{Proc} \rightarrow \langle \rangle]
\]
### Modeling channels

<table>
<thead>
<tr>
<th>Declaration (in PlusCal)</th>
<th>Declaration (in Distributed PlusCal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>`network=[p,q \in Proc</td>
<td>-&gt; ⟨⟩]`</td>
</tr>
</tbody>
</table>
# Modeling channels

## Declaration (in PlusCal)

\[
\text{network} = [p, q \in \text{Proc} | \rightarrow \langle \rangle]
\]

## Declaration (in Distributed PlusCal)

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

## Operation (in PlusCal)

```plaintext
macro mcast(p, m) {
    network := [s, d \in \text{Proc} | \rightarrow
    IF s = p \land d \neq p
    THEN Append(network[s,d], m)
    ELSE network[s,d]]
}
mcast(self, Request(clock));
```
## Modeling channels

<table>
<thead>
<tr>
<th>Declaration (in PlusCal)</th>
<th>Declaration (in Distributed PlusCal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>network=[p,q \in Proc</td>
<td>-&gt; ⟨⟩]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation (in PlusCal)</th>
<th>Operation (in Distributed PlusCal)</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{macro} mcast(p, m) {</td>
<td>* 1st argument: channel name</td>
</tr>
<tr>
<td>\hspace{1cm} network := [s,d \in Proc</td>
<td>-&gt;</td>
</tr>
<tr>
<td>\hspace{1cm} IF s = p /\ d # p</td>
<td>recipients and message</td>
</tr>
<tr>
<td>\hspace{1cm} THEN Append(network[s,d], m)</td>
<td>multicast(network,</td>
</tr>
<tr>
<td>\hspace{1cm} ELSE network[s,d]]</td>
<td>[self, p \in Proc</td>
</tr>
<tr>
<td>}</td>
<td>Request(clock)]);</td>
</tr>
<tr>
<td>mcast(self, Request(clock));</td>
<td></td>
</tr>
</tbody>
</table>
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 ∈ S ↦ e(x)])`
  - `multicast(ch, [x ∈ S ↦ e(x)])`
  - `clear(ch)`
Translation of Unordered Channels

\[ \text{channel } \langle id \rangle[\langle Expr_1 \rangle, \ldots, \langle Expr_N \rangle]; \]

▶ Translation based on TLA\(^+\) sets

▶ send(chan[e], msg) ≜

\[ \text{chan'} = \left[ \text{chan EXCEPT ![e] = chan[e] \cup \{msg\}} \right] \]

▶ receive(chan[e], var) ≜

\[ \exists \text{ temp } \in \text{chan[e]}: \]
\[ \land \text{ var'} = \text{temp} \]
\[ \land \text{ chan'} = \left[ \text{chan EXCEPT ![e] = chan[e] \setminus \{temp\}} \right] \]
Translation of FIFO Channels

\[ \text{fifo} \langle id \rangle \langle \text{Expr}_1, \ldots, \text{Expr}_N \rangle; \]

Translation based on TLA\(^+\) sequences

- send(chan[e], msg) ≜
  \[
  \text{chan'}' = \left[ \text{chan EXCEPT } ![e] = \text{Append}(\emptyset, \text{msg}) \right]
  \]

- receive(chan[e], var) ≜
  \[
  \begin{align*}
  &\text{\slash \slash } \text{Len}(\text{chan}[e]) > 0 \\
  &\text{\slash \slash } \text{var'} = \text{Head}(\text{chan}[e]) \\
  &\text{\slash \slash } \text{chan'} = \left[ \text{chan EXCEPT } ![e] = \text{Tail}(\emptyset) \right]
  \end{align*}
  \]
The variable $pc$ is indexed by processes and sub-processes

$$pc = \left[ \text{self \ in \ ProcSet} \right] \rightarrow$$

- CASE self \ in \ $P_i$ \ -> << $lbl_{i1}$, $lbl_{i2}$, ...>>
  - [] self \ in \ ...

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

Program counter
Translation to TLA+

\[
\text{exit: clock := clock + 1;}
\]
\[
\text{multicast(network, \{self, p \in Proc \setminus \{self\} \rightarrow Release(clock)\});}
\]

\[
\text{exit(self) ==}
\]
\[
\land pc[self][1] = "exit"
\land clock' = \{\text{clock EXCEPT ![self] = clock[self] + 1}\}
\land network' = \{\text{<<slf, p>> \in DOMAIN network \rightarrow}
\text{IF slf = self /\ p \in Proc \setminus \{self\} THEN Append(network[slf, p], Release(clock'[self]))
\text{ELSE network[slf, p]]}
\land pc' = \{\text{pc EXCEPT ![self][1] = "ncs"}\}
\land \text{UNCHANGED << req, ack, sndr, msg >>}
\]
## Contributions and future work

### Contributions

- An extension of PlusCal offering the possibility to define
  - Sub-Processes
  - Communication Channels
- A backward compatible translator to TLA$^+$
  
  https://github.com/hebaalkayed/DistributedPlusCal

### Future Work

- Introduce more types of communication channels
- Consider defining channel operations in a TLA$^+$ module