Formal Verification of Distributed Algorithms using Distributed PlusCal

Stephan Merz, Horatui Cirstea, Heba Al-kayed
Loria, Inria

I. Motivations

DISTRIBUTED systems are based on continuous interactions among components, these interactions produce bugs that are difficult to find by testing as they tend to be non-reproducible or not covered by test-cases.

Formal verification methods like model checking [?] depend on the users ability to specify the concerned system, TLA+ [?] is a formal language used to describe algorithms, it provides a flexibility and an expressiveness that enables it to specify and verify complicated algorithms concisely.

TLA+ relies on mathematical logic and formulas for structuring specifications whereas The PlusCal language [?] provides a simple pseudo-code like interface for the user to express concurrent systems. It maintains the expressiveness of TLA+ while providing the user with a more familiar syntax. Although PlusCal is very useful, when it comes to distributed algorithms it may enforce some limitation that make it difficult to express them in a natural way. As an example, distributed algorithms usually need to model several sub-processes coexisting and communicating as a part of a distributed node, PlusCal processes must all be declared at top level and cannot be easily used to create what the algorithm is aiming for.

II. Distributed PlusCal

DISTRIBUTED PLUSCAL is a language that extends PlusCal and is used to describe distributed algorithms.

A. Structure of an algorithm

The general structure and organization of a Distributed PlusCal algorithm is shown in Figure ??.

The `PlusCal options section` holds options to be passed to the translator, we notify the translator to parse a Distributed PlusCal algorithm by passing the $-distpcal$ option.

B. Communication Channels

The `declarations section` in Figure ?? allows the user to declare primitive constructs such as non-ordered channels and FIFO channels in addition to PlusCal variables.

The general structure for an unordered channel declaration is shown below. An unordered channel is declared with the keyword **channel** or **channels**.

$$\textbf{channel}\ \langle id\rangle[\langle dimension1, ..., dimensionN\rangle];$$

```
(* 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>
*)
```

Fig. 1. General structure of a Distributed PlusCal algorithm

The corresponding TLA+ translation of the unordered channel declared with N dimensions is shown below.

$$\langle id\rangle = [\langle d1 \in dimension1, ...dN \in dimensionN| -> \{\}\rangle];$$

A FIFO channel is declared with the keyword **fifo** or **fifos**.

$$\textbf{fifo}\ \langle id\rangle[\langle dimension1, ..., dimensionN\rangle];$$

The TLA+ translation a FIFO channel is as follows

$$\langle id\rangle = [\langle d1 \in dimension1, ...dN \in dimensionN| -> \langle\rangle\rangle];$$

The $dimensions$ are TLA+ sets that are defined in the TLA+ file, they represent the keys of the channel. Defining a multidimensional channel corresponds to defining a set of channels.

The supported channel operators include `send`, `receive`, `broadcast`, `multicast`, `clear`.

C. Sub-Processes

In the `Process Section` in Figure ?? each process can hold multiple sub-processes each with its own body of statements. This enables the process to execute multiple tasks in parallel.

The listing below shows a small part of an example we developed for modeling the two phase commit protocol using Distributed PlusCal along the TLA$^+$ translation produced by our translator.

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

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

The value of the pc variable in PlusCal is a single string equal to the label of the next statement to be executed with respect to a process. In Distributed PlusCal we extended the definition to indicate which sub-process is involved as well. In the listing above at line 2 is the pc variable references the 2nd sub-process.

III. Evalation

In distributed algorithms several processes or threads may coexist and asynchronously communicate while being a part of a distributed node. Distributed PlusCal firstly introduced sub-processes to allow the declaration of these threads, and secondly we provided the user with communication channel constructs to assist in the communication between those processes.

Extending the PlusCal translator allowed our translator to be backward compatible. The Distributed PlusCal translator can translate PlusCal models as well as Distributed PlusCal models.