Process Communication

Zunaira Zaman

May 10, 2022

1 Single Threads and Replicated Threads implementation per Process

In this study, we are proposing the syntax to have single threads and replicated threads for a process. In the previous work on the Distributed PlusCal, we were allowed to create multiple singly-threads per process, an enhancement to this process communication semantics is to have multiple single and replicated threads per process.

The syntax for the main process will remain same as in pluscal, for the single thread implementation, we will be utilizing the sub-process syntax from the distributed Pluscal and replicated threads can be created as follows in the code example. Both single threads and replicated threads can access the variables declared for the main processs, which makes the communication easier between threads.

```
(*--algorithm Syntax
variables x = 0;
process p \in Processes  //main process
begin
    A: x := x+2 ;
end process
begin                    //Single Thread
   B: x := x+1;
end subprocess
begin sub \in [2..3]    //Replicated Thread
   C: x := x+1;
end subprocess


process q \in ProcessesQ //main Process
variables y = 0;
begin
    D: y := y + 2 ;
end process
begin                  //Single Thread
   E: y := x * y;
end subprocess
begin                  //Single Thread
   F: y := y+2;
end subprocess
begin sub \in [4..5]  //Replicated Thread
   variables z=0;
   G: y := y-1;
      z := z + 1;
end subprocess
end algorithm;
*)
```

2 TLA+ Translation

Though the structure for single threads remains same as it was in Distributed Pluscal. But since, we are introducing replicated threads as an enhancement feature in the current process implementation. We are defining new structures and sets for the translation of the above code implementation.

```
ProcSet == (ProcessesP) \cup (ProcessesQ)
Procs1 == {<<p>>: p \in ProcessesP} //main process
Procs1SingleThreads1 == {<<p,1>>: p \in ProcessesP} //single Thread
Procs1ReplicatedThreads2 == {<<p, 2, i>>: p \in ProcessesP, i \in 2..3 } //Replicated Thread
```

2.1 Main Processes TLA+ Translation

In the TLA+ translation, we have proposed to generate Procs1 set for a single independent main process. In case, there are multiple independent processes in the program, then TLA+ shall be able to generate multiple Procs sets for each process. For instance, if we have two main processes such as ProcessesP and ProcessesQ in the ProcSet. It will generate further two procs1 and procs2 sets for the main processes respectively.

```
ProcSet == (ProcessesP) \cup (ProcessesQ)
Procs1 == {<<p>>: p \in ProcessesP} //main process P
Procs2 == {<<p>>: p \in ProcessesQ} //main process Q
```

2.2 Single Threads TLA+ Translations

Similarly, in the following case program, we have two single processes in the ProcessesQ. The TLA+ translation should be able to identify each single thread separately and it should generate the number of Procs1SingleThread1 depending on the number of single threads needed in the program.

```
//TLA+ Translation for single threads
Procs2 == {<<p>>: p \in ProcessesQ}
Procs2SingleThreads1 == {<<p,1>>: p \in ProcessesQ}
Procs2SingleThreads2 == {<<p,2>>: p \in ProcessesQ}
```

2.3 Replicated Threads TLA+ Translation

Since, we have introduced new structure to enable users to write replicated threads in Distributed PlusCal as shown in the figure:

```
//replicated Threads structure using p-syntax
begin sub \in [2..3]
   C: x := x+1;
end subprocess
```

Now, we need to introduce the TLA+ translation semantics for replicated threads. If a process has a replicated thread(s), it should generate the Procs1ReplicatedThreads2 set(s). The TLA+ translation will generate the Procs1ReplicatedThread set(s) depending on the number of replicated threads we have inside an independent process.

```
//TLA+ Translation for the Replicated Threads
Procs1ReplicatedThreads2 == {<<p, 2, i>>: p \in Processes, i \in 2..3 }
```

2.3.1 PC definition

The pc variable value in PlusCal is a value which represents a single string equal to the label of the next statement to be executed with respect to a process. Further in the distributed PlusCal, this PC

variable value has been extended to include which subprocess is involved. In our case, we have changed this implementation to include process, single threads, and replicated threads.

```
Init == (* Global variables *)
       /\ x = [self \in Processes |-> 0]
       /\ y = [self \in ProcessesQ |-> 0]
       /\ pc = [ p \in Procs1 \union Procs2 \union Procs1SingleThreads1
                   \union Procs1ReplicatedThreads2
                   \union Procs2SingleThreads1
                   \union Procs2SingleThreads2
                   \union Procs2ReplicatedThreads3
                 |-> CASE p \in Procs1 -> "A"
                 [] p \in Procs1SingleThreads1 -> "B"
                 [] p \in Procs1ReplicatedThreads2 -> "C"
                 [] p \in Procs2 -> "D"
                 [] p \in Procs2SingleThreads1 -> "E"
                 [] p \in Procs2SingleThreads2 -> "F"
                 [] p \in Procs2ReplicatedThreads3 -> "G" ]
```

2.3.2 TLA+ Translation for Processes, Single threads and Replicated Threads

```
A(self) == /\ pc[<<self>>] = "A"
          /\ x' = [x EXCEPT ![self] = x[self] + 2]
          /\ pc' = [pc EXCEPT ![<<self>>] = "Done"]
          /\ y' = y
          /\ z' = z

B(self) == /\ pc[<<self, 1>>] = "B"
          /\ x' = [x EXCEPT ![self] = x[self] + 1]
          /\ pc' = [pc EXCEPT ![<<self, 1>>] = "Done"]
          /\ y' = y
          /\ z' = z
C(self, i) == /\ pc[<<self, 2, i>>] = "C"
          /\ x' = [x EXCEPT ![self] = x[self] + 1]
          /\ pc' = [pc EXCEPT ![<<self, 2, i>>] = "Done"]
          /\ y' = y
          /\ z' = z

p(self) == A(self)
pthread1(self) == B(self)
pthread2(self,i) == C(self,i)


D(self) == /\ pc[<<self>>] = "D"
          /\ y' = [y EXCEPT ![self] = y[self] + 2]
          /\ pc' = [pc EXCEPT ![<<self>>]= "Done"]
          /\ x' = x
          /\ z' = z

E(self) == /\ pc[<<self, 1>>] = "E"
          /\ y' = [y EXCEPT ![self] = x[self] * y[self]]
          /\ pc' = [pc EXCEPT ![<<self, 1>>] = "Done"]
          /\ x' = x
          /\ z' = z
F(self) == /\ pc[<<self, 2>>] = "F"
          /\ y' = [y EXCEPT ![self] = y[self] + 2]
          /\ pc' = [pc EXCEPT ![<<self, 1>>] = "Done"]
          /\ x' = x
```

```
          /\ z' = z
G(self, i) == /\ pc[<<self, 3, i>>] = "G"
              /\ y' = [y EXCEPT ![self] = y[self] - 1]
              /\ z' = [z EXCEPT ![self] = z[self] + 1]
              /\ pc' = [pc EXCEPT ![<<self, 2, i>>] = "Done"]
              /\ x'= x


q(self) == D(self)

qthread1(self) == E(self)
qthread2(self) == F(self)
qthread3(self,i) == G(self,i)
```

2.3.3 Next TLA+ Translation

For the next definition, it will simply accumulate all the processes, and their respective single threads and multiple threads.

```
Next ==  \/ \E self \in Processes : p(self)
         \/ \E self \in Processes : pthread1(self)
         \/ \E self \in Processes, i \in 2..3 : pthread2(self, i)
         \/ \E self \in ProcessesQ : q(self)
         \/ \E self \in ProcessesQ : qthread1(self)
         \/ \E self \in ProcessesQ : qthread2(self)
         \/ \E self \in ProcessesQ, i \in 4..5 : qthread3(self, i)
```