A Choreographic Language for PRISM

... Author: Please enter affiliation as second parameter of the author macro

... Author: Please enter affiliation as second parameter of the author macro

Abstract

This is the abstract

2012 ACM Subject Classification Theory of computation → Type theory; Computing methodologies → Distributed programming languages; Theory of computation → Program verification

Keywords and phrases Session types, PRISM, Model Checking

Digital Object Identifier 10.4230/LIPIcs.ITP.2023.m

Funding This work was supported by

1 Formal Language

In this section, we provide the formal definition of our choreographic language as well as process algebra representing PRISM [?].

1.1 PRISM

We start by describing PRISM semantics. Except from transforming some informal text in precise rules, Our formalisation closely follows that found on the PRISM website [?].

Syntax. Let p range over a (possibly infinite) set of module names $\mathcal{R}$, a over a (possibly infinite) set of labels $\mathcal{L}$, x over a (possibly infinite) set of variables Var, and v over a (possibly infinite) set of values Val. Then, the syntax of PRISM is given by the following grammar:

$$
\begin{array}{llcll}
\text{(Networks)} & N, M & ::= & \mathbf{0} & \text{empty network} \\
 & & | & \mathsf{p} : \{F_i\}_i & \text{module} \\
 & & | & M|[A]|M & \text{parallel composition} \\
 & & | & M/A & \text{action hiding} \\
 & & | & \sigma M & \text{substitution} \\
\\
\text{(Commands)} & F & ::= & [a]g \to \Sigma_{i\in I}\{\lambda_i : u_i\} & g \text{ is a boolean expression in } E \\
\\
\text{(Assignment)} & u & ::= & (x' = E) & \text{update } x\text{, element of } \mathcal{V}\text{, with } E \\
 & & | & A\&A & \text{multiple assignments} \\
\text{(Expr)} & E & ::= & f(\tilde{E}) \quad | \quad x \quad | \quad v &
\end{array}
$$

Networks are the top syntactic category for system of modules composed together. The term $CEnd$ represent an empty network. A module $\mathsf{p} : \{F_i\}_i$ is identified by its name p and a set of commands F_i. Networks can be composed in parallel, in a CSP style: a term like $M_1|[A]|M_2$ says that networks M_1 and M_2 can interact with each other using labels in the finite set A. The term M/A is the standard CSP/CCS hiding operator. Finally σM is equivalent to applying the substitution σ to all variables in x. A substitution is a function that given a variable returns a value. When we write σN we refer to the term obtained by replacing every free variable x in N with $\sigma(x)$. Marco: Is this really the way substitution is used? Where does it become important?

Semantics. In order to give a probabilistic semantics to PRISM, we proceed by steps. First, we define $\{\![-]\!\}$, as the closure of the following rules:

$$\frac{}{F_i \in \{\![\mathsf{p} : \{F_i\}_i]\!\}}\ (\mathsf{Module}) \qquad \frac{[]E \to \{\lambda_i : x_i = E_i\}_{i\in I} \in \{\![M_j]\!\} \quad j \in \{1,2\}}{[]E \to \{\lambda_i : x_i = E_i\}_{i\in I} \in \{\![M_1|[A]|M_2]\!\}}\ (\mathsf{Par}_1)$$

$$\frac{[a]E \to \{\lambda_i : x_i = E_i\}_{i\in I} \in \{\![M_j]\!\} \quad a \notin A \quad j \in \{1,2\}}{[a]E \to \{\lambda_i : x_i = E_i\}_{i\in I} \in \{\![M_1|[A]|M_2]\!\}}\ (\mathsf{Par}_2)$$

$$\frac{[a]E \to \{\lambda_i : x_i = E_i\}_{i\in I} \in \{\![M_1]\!\} \quad [a]E' \to \{\lambda'_j : y_j = E'_j\}_{j\in J} \in \{\![M_2]\!\} \quad a \in A}{[]E \wedge E' \to \{\lambda_i * \lambda'_j : x_i = E_i \wedge y_j = E'_j\}_{i\in I, j\in J} \in \{\![M_1|[A]|M_2]\!\}}\ (\mathsf{Par}_3)$$

$$\frac{[]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M]\!\}}{[]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M/A]\!\}}\ (\mathsf{Hide}_1) \qquad \frac{[a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M]\!\} \quad a \notin A}{[a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M/A]\!\}}\ (\mathsf{Hide}_2)$$

$$\frac{[a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M]\!\} \quad a \in A}{[]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M/A]\!\}}\ (\mathsf{Hide}_3) \qquad \frac{[]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M]\!\}}{[]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![\sigma M]\!\}}\ (\mathsf{Subst}_1)$$

$$\frac{[a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M]\!\} \quad a \notin \mathsf{dom}(\sigma)}{[a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![\sigma M]\!\}}\ (\mathsf{Subst}_2)$$

$$\frac{[a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![M]\!\} \quad a \in \mathsf{dom}(\sigma)}{[\sigma a]E \to \{\lambda_j : x_i = E_i\}_{i\in I} \in \{\![\sigma M]\!\}}\ (\mathsf{Subst}_3)$$

The rules above work with modules, parallel composition, name hiding, and substitution. The idea is that given a network, we wish to collect all those commands F that are contained in the network, independently from which module they are being executed in. Intuitively, we can regard $\{\![N]\!\}$ as a set, where starting from all commands present in the syntax, we do some filtering and renaming, based on the structure of the network.

Now, given $\{\![N]\!\}$, we define a transition system that shows how the system evolves. In order to do so, let state be a function that given a variable in Var returns a value in Val. Then, given an initial state state_0, we can define a transition system where each of node is a (different) state function. Then, we can move from state_1 to state_2 whenever

That means that ones we have a set of executable rules, we can start building a transition system. In order to do so, we

$$W(M) = \{F \mid F \in \{\![M]\!\}\}$$

$$X = \{x_1, \ldots, x_n\}$$

$$\sigma : X \to V$$

1.2 Choreographies

Syntax. Our choreographic language is defined by the following syntax:

$$(\text{Chor}) \quad C \ ::= \ \{\mathsf{p}_i\}_{i\in I} \ +\{\lambda_j : x_j = E_j;\ C_j\}_{j\in J} \ \mid \ \mathsf{if}\ E@\{\mathsf{p}_i\}_{i\in I}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2 \ \mid \ X \ \mid \ \mathbf{0}$$

We briefly comment the various constructs. The syntactic category C denotes choreographic programmes. The term $\{\mathsf{p}_i\}_{i\in I} \ +\{\lambda_j : x_j = E_j;\ C_j\}_{j\in J}$ denotes an interaction between the roles p_i. The value λ_j is a real number representing the rate. ...

1.3 Projection from Choreographies to PRISM

Mapping Choreographies to PRISM. We need to run some standard static checks because, since there is branching, some terms may not be projectable.

$$f : C \longrightarrow \texttt{network} \longrightarrow \texttt{network} \qquad \texttt{network} : \mathcal{R} \longrightarrow \mathrm{Set}(F)$$

$$f\Big(\ \mathsf{p}_1 \longrightarrow \{\mathsf{p}_i\}_{i\in I} \oplus \{[\lambda_j]x_j = E_j : D_j\}_{j\in J}, \texttt{network}\Big)$$

$$=$$

$$\begin{array}{l}
\mathsf{label} = \texttt{newlabel}();\\
\mathsf{for}\ \mathsf{p}_k \in \mathbf{roles}\{\\
\quad \mathsf{for}\ j \in J\{\\
\quad\quad \texttt{network} = \texttt{add}(\mathsf{p}_k, [\mathsf{label}]s_{\mathsf{p}_k} = \texttt{state}(\mathsf{p}_k) \rightarrow\ \lambda_j : x_j = E_j\ \&\ s'_{\mathsf{p}_k} = \texttt{genNewState}(\mathsf{p}_k));\\
\quad \}\\
\}\\
\mathsf{for}\ j \in J\{\\
\quad \texttt{network} = f(D_j, \texttt{network});\\
\}\\
\mathsf{return}\ \texttt{network}
\end{array}$$

$$f\Big(\ \mathsf{if}\ E@\{\mathsf{p}_i\}_{i\in I}\ \mathsf{then}\ C_1\ \mathsf{else}\ C_2, \texttt{network}\Big)$$

$$=$$

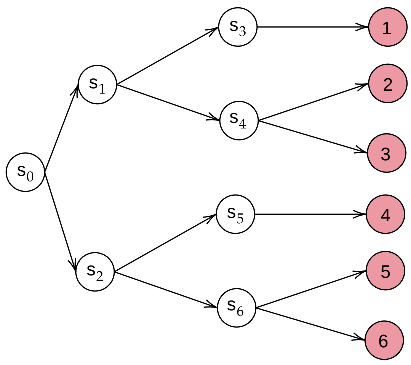
$$\begin{array}{l}
\mathsf{for}\ \mathsf{p}_k \in \mathbf{roles}\{\\
\quad \texttt{network} = \texttt{add}(\mathsf{p}_k, [\,]s_{\mathsf{p}_k} = \texttt{state}(\mathsf{p}_k)\ \&\ f(E));\\
\quad \texttt{network} = f(C_1, \texttt{network});\\
\quad \texttt{network} = f(C_2, \texttt{network});\\
\}\\
\text{return}\ \texttt{network}
\end{array}$$

2 Tests

We tested our language by various examples.

2.1 The Dice Program

The first example we present is the Dice Program[1] [4]. The following program models a die using only fair coins. Starting at the root vertex (state 0), one repeatedly tosses a coin. Every time heads appears, one takes the upper branch and when tails appears, the lower branch. This continues until the value of the die is decided.



We modelled the program using the choreographic language (Listing 1) and we were able to generate the corresponding PRISM program, reported in Listing 2.

```
preamble
"dtmc"
endpreamble

n = 1;
Dice → Dice : "d : [0..6] init 0;" ;

{
DiceProtocol0 ≔ Dice → Dice : (+["0.5*1"] " "&&" " . DiceProtocol1
                                +["0.5*1"] " "&&" " . DiceProtocol2)

DiceProtocol1 ≔ Dice → Dice : (+["0.5*1"] " "&&" " .
                      Dice → Dice : (+["0.5*1"] " "&&" " . DiceProtocol1
                                     +["0.5*1"] "(d'=1)"&&" " . DiceProtocol3)
                                +["0.5*1"] " "&&" " .
                      Dice → Dice : (+["0.5*1"] "(d'=2)"&&" " . DiceProtocol3
                                     +["0.5*1"] "(d'=3)"&&" " . DiceProtocol3))

DiceProtocol2 ≔ Dice → Dice : (+["0.5*1"] " "&&" " .
                      Dice → Dice : (+["0.5*1"] " "&&" " . DiceProtocol2
                                     +["0.5*1"] "(d'=4)"&&" " . DiceProtocol3)
                               +["0.5*1"] " "&&" " .
                      Dice → Dice : (+["0.5*1"] "(d'=5)"&&" " . DiceProtocol3
                                     +["0.5*1"] "(d'=6)"&&" " . DiceProtocol3))
```

[1] https://www.prismmodelchecker.org/casestudies/dice.php

```
DiceProtocol₃ ≔ Dice → Dice : (["1*1"] " "&&" ".DiceProtocol₃)
}
```

Listing 1 Choreographic language for the Dice Program.

```
dtmc

module Dice
       Dice : [0..11] init 0;
       d : [0..6] init 0;

       [] (Dice=0) → 0.5 : (Dice'=2) + 0.5 : (Dice'=6);
       [] (Dice=2) → 0.5 : (Dice'=3) + 0.5 : (Dice'=4);
       [] (Dice=3) → 0.5 : (Dice'=2) + 0.5 : (d'=1)&(Dice'=10);
       [] (Dice=4) → 0.5 : (d'=2)&(Dice'=10) + 0.5 : (d'=3)&(Dice'=10);
       [] (Dice=6) → 0.5 : (Dice'=7) + 0.5 : (Dice'=8);
       [] (Dice=7) → 0.5 : (Dice'=6) + 0.5 : (d'=4)&(Dice'=10);
       [] (Dice=8) → 0.5 : (d'=5)&(Dice'=10) + 0.5 : (d'=6)&(Dice'=10);
       [] (Dice=10) → 1 : (Dice'=10);

endmodule
```

Listing 2 Generated PRISM program for the Dice Program.

By comparing our model with the one presented in the PRISM documentation, we noticed that the difference is the number assumed by the variable `Dice`. In particular, the variable does not assume the values 1, 5 and 9. This is due to how the generation in presence of a branch is done. However, this does not cause any problems since the updates are done correctly. Moreover, to prove the generated program is correct, we show that the probability of reaching a state where

$$\texttt{d=k} \text{ for } \texttt{k} = 1, \ldots, 6 \text{ is } 1/6.$$

The results are displayed in Figure 1, where also the results obtained with the original PRISM model are shown.

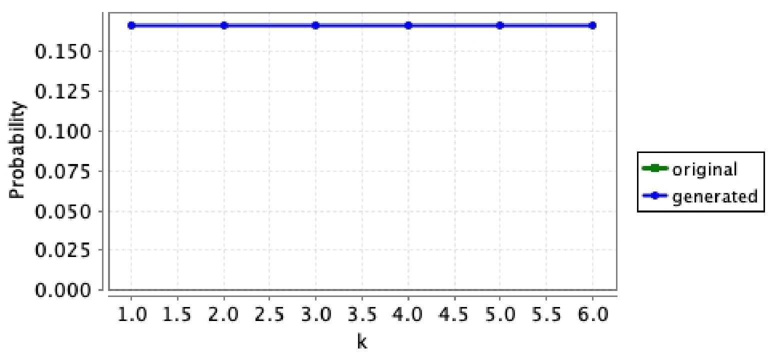


Figure 1 Probability of reaching a state where $d = k$, for $k = 1, \ldots, 6$.

2.2 Simple Peer-To-Peer Protocol

This case study describes a simple peer-to-peer protocol based on BitTorrent[2]. The model comprises a set of clients trying to download a file that has been partitioned into K blocks. Initially, there is one client that has already obtained all of the blocks and N additional clients with no blocks. Each client can download a block from any of the others but they can only attempt four concurrent downloads for each block.

The code we analyze with $k = 5$ and $N = 4$ is reported in Listing 3.

```
preamble
"ctmc"
"const double mu=2;"
"formula rate1=mu*(1+min(3,b11+b21+b31+b41));"
"formula rate2=mu*(1+min(3,b12+b22+b32+b42));"
"formula rate3=mu*(1+min(3,b13+b23+b33+b43));"
"formula rate4=mu*(1+min(3,b14+b24+b34+b44));"
"formula rate5=mu*(1+min(3,b15+b25+b35+b45));"
endpreamble

n = 4;
n = 4;

Client[i] → i in [1...n]
Client[i] : "b[i]1 : [0..1];", "b[i]2 : [0..1];", "b[i]3 : [0..1];", "b[i]4 :
    [0..1];", "b[i]5 : [0..1];" ;

{
PeerToPeer := Client[i] → Client[i]:
                     (+["rate1*1"] "(b[i]1'=1)"&&" " . PeerToPeer
                      +["rate2*1"] "(b[i]2'=1)"&&" " . PeerToPeer
                      +["rate3*1"] "(b[i]3'=1)"&&" " . PeerToPeer
                      +["rate4*1"] "(b[i]4'=1)"&&" " . PeerToPeer
                      +["rate5*1"] "(b[i]5'=1)"&&" " . PeerToPeer)
}
```

Listing 3 Choreographic language for the Peer-To-Peer Protocol.

Part of the generated PRISM code is shown in Listing 4 and it is faithful with what reported in the PRISM documentation.

```
ctmc
const double mu=2;
formula rate1=mu*(1+min(3,b11+b21+b31+b41));
formula rate2=mu*(1+min(3,b12+b22+b32+b42));
formula rate3=mu*(1+min(3,b13+b23+b33+b43));
formula rate4=mu*(1+min(3,b14+b24+b34+b44));
formula rate5=mu*(1+min(3,b15+b25+b35+b45));

module Client1
       Client1 : [0..1] init 0;
       b11 : [0..1];
       b12 : [0..1];
       b13 : [0..1];
```

[2] https://www.prismmodelchecker.org/casestudies/peer2peer.php

```
        b14 : [0..1];
        b15 : [0..1];

        [] (Client1=0) → rate1 : (b11'=1)&(Client1'=0);
        [] (Client1=0) → rate2 : (b12'=1)&(Client1'=0);
        [] (Client1=0) → rate3 : (b13'=1)&(Client1'=0);
        [] (Client1=0) → rate4 : (b14'=1)&(Client1'=0);
        [] (Client1=0) → rate5 : (b15'=1)&(Client1'=0);

endmodule
```

Listing 4 Generated PRISM program for the Peer-To-Peer Protocol.

In Figure 2, we compare the values obtained for the probability that all clients have received all blocks by time $0 \leq T \leq 1.5$ both for our generated model and the model reported in the documentation.

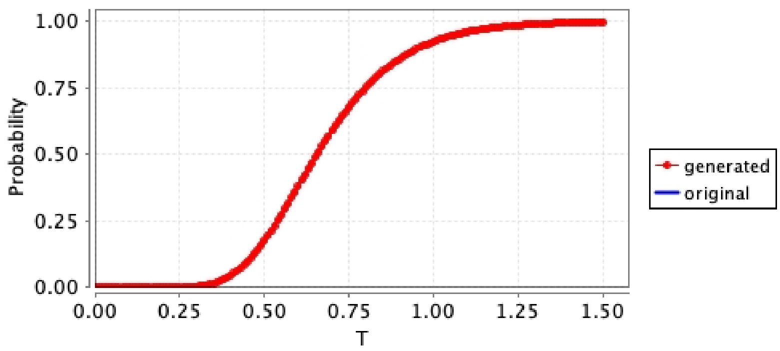


Figure 2 Probability that clients received all the block before T, with $0 \leq T \leq 1.5$.

2.3 Proof of Work Bitcoin Protocol

This protocol represents the Proof of Work implemented in the Bitcoin blockchain. In[2], a Bitcoin system is the result of the parallel composition of n Miner processes, n *Hasher* processes and a process called *Network*. *Hasher* processes model the attempts of the miners to solve the cryptopuzzle, while the *Network* process model the broadcast communication among miners. We tested our system by considering a protocol with $n = 5$ miners and it is reported in Listing 5.

```
preamble
"ctmc"
"const T"
"const double r = 1;"
"const double mR = 1/600;"
"const double lR = 1-mR;"
"const double hR1 = 0.25;"
"const double hR2 = 0.25;"
"const double hR3 = 0.25;"
"const double hR4 = 0.25;"
"const double rB = 1/12.6;"
"const int N = 100;"
endpreamble
```

```
n = 4;

Hasher[i] -> i in [1...n] ;

Miner[i] -> i in [1...n]
Miner[i] : "b[i] : block {m[i],0;genesis,0} ;", "B[i] : blockchain [{genesis,0;
    genesis,0}];" ,"c[i] : [0..N] init 0;", "setMiner[i] : list [];" ;

Network ->
Network : "set1 : list [];", "set2 : list [];", "set3 : list [];" , "set4 : list
    [];";

{
PoW := Hasher[i] → Miner[i] :
(+["mR*hR[i]"] " "&&"(b[i]'=createB(b[i],B[i],c[i]))&(c[i]'=c[i]+1)" .
       Miner[i] → Network :
              (["rB*1"] "(B[i]'=addBlock(B[i],b[i]))" &&
              foreach(k != i) "(set[k]'=addBlockSet(set[k],b[i]))" @Network .PoW)
 +["lR*hR[i]"] " " && " " .
       if "!isEmpty(set[i])"@Miner[i] then {
              ["r"] "(b[i]'=extractBlock(set[i]))"@Miner[i] .
                     Miner[i] → Network :
                     (["1*1"] "(setMiner[i]' = addBlockSet(setMiner[i] , b[i]))"
                         &&"(set[i]' = removeBlock(set[i],b[i]))" . PoW)
       }
       else{
              if "canBeInserted(B[i],b[i])"@Miner[i] then {
                     ["1"] "(B[i]'=addBlock(B[i],b[i]))
                     &(setMiner[i]'=removeBlock(setMiner[i],b[i]))"@Miner[i] . Pow
              }
              else{
                     PoW
              }
       }
)
}
```

Listing 5 Choreographic language for the Proof of Work Bitcoin Protocol.

Part of the generated PRISM code is shown in Listing 6.

```
ctmc
const T;
const double r = 1;
const double mR = 1/600;
const double lR = 1-mR;
const double hR1 = 0.25;
const double hR2 = 0.25;
const double hR3 = 0.25;
const double hR4 = 0.25;
const double rB = 1/12.6;
const int N = 100;

module Miner1
Miner1 : [0..7] init 0;
```

```
b1 : block {m1,0;genesis,0} ;
B1 : blockchain [{genesis,0;genesis,0}];
c1 : [0..N] init 0;
setMiner1 : list [];

[PZKYT] (Miner1=0) → hR1 : (b1'=createB(b1,B1,c1))&(c1'=c1+1)&(Miner1'=1);
[EUBVP] (Miner1=0) → hR1 : (Miner1'=2);
[HXYKO] (Miner1=1) → 1 : (B1'=addBlock(B1,b1))&(Miner1'=0);
[] (Miner1=2)&!isEmpty(set1) → r : (b1'=extractBlock(set1))&(Miner1'=4);
[SRKSV] (Miner1=4) → 1 : (setMiner1' = addBlockSet(setMiner1 , b1))&(Miner1'=0);
[] (Miner1=2)&!(!isEmpty(set1)) → 1 : (Miner1'=5);
[] (Miner1=5)&canBeInserted(B1,b1) → 1 : (B1'=addBlock(B1,b1))
              &(setMiner1'=removeBlock(setMiner1,b1))&(Miner1'=0);
[] (Miner1=5)&!(canBeInserted(B1,b1)) → 1 : (Miner1'=0);
endmodule
...
module Network
Network : [0..1] init 0;
set1 : list [];
...

[HXYKO] (Network=0) → 1 : (set2'=addBlockSet(set2,b2))&(set3'=addBlockSet(set3,b3
    ))&(set4'=addBlockSet(set4,b4))&(Network'=0);
[SRKSV] (Network=0) → 1 : (set1' = removeBlock(set1,b1))&(Network'=0);
...

endmodule

module Hasher1
Hasher1 : [0..1] init 0;

[PZKYT] (Hasher1=0) → mR : (Hasher1'=0);
[EUBVP] (Hasher1=0) → lR : (Hasher1'=0);

endmodule
```

Listing 6 Generated PRISM program for the Peer-To-Peer Protocol.

In Figure 3, we compare the values obtained for the probability that at least one miner has mined a block both for the generated model and the model presented in [2].

2.4 Random Graphs Protocol

In this case study[3] we investigate the likelihood that a pair of nodes are connected in a random graph. More precisely, we take into account the the set of random graphs $G(n, p)$, i.e. the set of random graphs with n nodes where the probability of there being an edge between any two nodes equals p.

```
       preamble
"mdp"
"const double p;"
endpreamble
```

[3] https://www.prismmodelchecker.org/casestudies/graph_connected.php

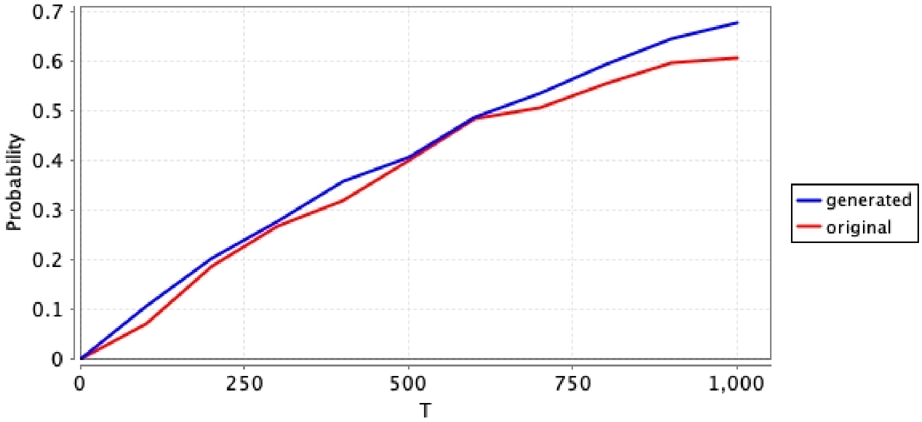


Figure 3 Probability at least one miner has created a block.

```
n = 3;

PC ->
PC : " ";

M[i] -> i in [1...n]
Module[i] : "varM[i] : bool;";

P[i] -> i in [1...n]
P[i] : "varP[i] : bool;";

{
GraphConnected0 :=
        PC -> M[i] : (+["1*p"] " "&&"(varM[i]'=true)". END
                        +["1*(1-p)"] " "&&"(varM[i]'=false)". END)
        PC -> P[i] : (+["1*p"] " "&&"(varP[i]'=true)" . END
                        +["1*(1-p)"] " "&&"(varP[i]'=false)".
                        if "(PC=6)&!varP[i]&((varP[i] & varM[i]) | (varM[i+1] & varP[
                            i+2])) "@P[i] then {
                                    ["1"]"(varP[i]'=true)"@P[i] . GraphConnected0
                        })
}
```

Listing 7 Choreographic language for the Random Graphs Protocol.

The model is divided in two parts: at the beginning the random graph is built. Then they find nodes that have a path to node 2 by searching for nodes for which one can reach (in one step) a node for which they have already found the existence of a path to node 2. Part of the generated PRISM code is shown in Listing 8 (we do not report modules M2, M3, P2, P3).

```
mdp
const double p;

module PC
   PC : [0..7] init 0;

```

```
    [DPPGR] (PC=0) → 1 : (PC'=1);
    [YCJJG] (PC=1) → 1 : (PC'=2);
    [TWGVA] (PC=2) → 1 : (PC'=3);
    [NODPZ] (PC=3) → 1 : (PC'=4);
    [FDALJ] (PC=4) → 1 : (PC'=5);
    [DCKXC] (PC=5) → 1 : (PC'=6);
endmodule

module M1
    M1 : [0..1] init 0;
    varM1 : bool;

    [DPPGR] (M1=0) → p :(varM1'=true)&(M1'=0) + (1-p) :(varM1'=false)&(M1'=0);
endmodule

...

module P1
    P1 : [0..3] init 0;
    varP1 : bool;

    [NODPZ] (P1=0) → p:(varP1'=true)&(P1'=0) + (1-p):(varP1'=false)&(P1'=0);
    [] (P1=0)&(PC=6)&!varP1&((varP1 & varM1) | (varM2& varP3))
                                    → 1 : (varP1'=true)&(P1'=0);
endmodule
```

Listing 8 Generated PRISM program for the Random Graphs Protocol.

The model is very similar to the one presented in the PRISM repository, the main difference is that we use state variables also for the modules $\mathtt{P}_i$ and $\mathtt{M}_i$.

In Figure 4, we compare the results obtained with the two models.

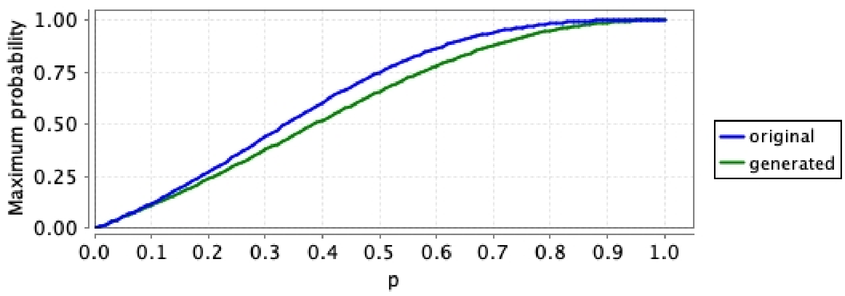


Figure 4 Probability that the nodes 1 and 2 are connected.

2.5 Hybrid Casper Protocol

The last case we study is the Hybrid Casper Protocol presented in [3]. The protocol models what happened in the Ethereum blockchain while it was implemented the hybrid Casper protocol: an hybrid protocol that includes features of the Proof of Work and the Proof of Stake protocols. The modeling language is reported in Listing 9 while (part of) the generated PRISM code can be found in Listing 8.

```
preamble
```

```
"ctmc"
"const int EpochSize = 2;"
"const k = 1;"
"const double rMw = 1/12.6;"
"const epochs = 0;"
"const double T;"
"const int N = 100;"
"const double rC = 1/(14*EpochSize);"
"const double mR =1/14;"
"const double lR = 10;"
endpreamble

n = 5;

Validator[i] -> i in [1...n]
Validator[i] : "b[i] : block {m[i],0;genesis,0};", "lastJ[i] : block {m[i],0;
    genesis,0};", "L[i] : blockchain [{genesis,0;genesis,0}];", "c[i] : [0..N]
    init 0;", "setMiner[i] : list [];", "heightCheckpoint[i] : [0..N] init 0;", "
    heightLast[i] : [0..N] init 0;", "lastFinalized[i] : block {genesis,0;genesis
    ,0};", "lastJustified[i] : block {genesis,0;genesis,0};", "lastCheck[i] :
    block {genesis,0;genesis,0};", "votes[i] : [0..1000] init 0;", "
    listCheckpoints[i] : list [];";

Network ->
Network : "set1 : list [];", "set2 : list [];", "set3 : list [];" , "set4 : list
    [];" , "set5 : list [];";

Vote_Manager ->
Vote_Manager : "Votes : hash []; ", "tot_stake : [0..120000] init 50;", "stake1 :
    [0..N] init 10;", "stake2 : [0..N] init 10;", "stake3 : [0..N] init 10;", "
    stake4 : [0..N] init 10;", "stake5 : [0..N] init 10;";

{
PoS := Validator[i] -> Validator[i] :
       (+["mR*1"] "(b[i]'=createB(b[i],L[i],c[i]))&(c[i]'=c[i]+1)"&&" " .
           if "!(mod(getHeight(b[i]),EpochSize)=0)"@Validator[i] then{
                    Validator[i] -> Network : (["1*1"] "(L[i]'=addBlock(L[i],b[i
                        ]))" && foreach(k!=i) "(set[k]'=addBlockSet(set[k],b[i]))
                        "@Network .PoS)
              }
              else{
                    Validator[i] -> Network : (["1*1"] "(L[i]'=addBlock(L[i],b[i
                        ]))" && foreach(k!=i) "(set[k]'=addBlockSet(set[k],b[i]))
                        "@Network.
                    Validator[i] -> Vote_Manager :(["1*1"] " "&&"(Votes'=addVote(
                        Votes,b[i],stake[i]))".PoS))
              }
        +["lR*1"] " "&&" " .
              if "!isEmpty(set[i])"@Validator[i] then {
                   ["1"] "(b[i]'=extractBlock(set[i]))"@Validator[i] .
                          if "!canBeInserted(L[i],b[i])"@Validator[i] then {
                                 PoS
                          }
                          else{
                                 if "!(mod(getHeight(b[i]),EpochSize)=0)"
```

```
                                              @Validator[i] then {
                                                 Validator[i] -> Network : (["1*1"] "(
                                                     setMiner[i]' = addBlockSet(setMiner
                                                     [i] , b[i]))"&&"(set[i]' =
                                                     removeBlock(set[i],b[i]))" . PoS)
                                          }
                                          else{
                                                 Validator[i] -> Network : (["1*1"] "(
                                                     setMiner[i]' = addBlockSet(setMiner
                                                     [i] , b[i]))"&&"(set[i]' =
                                                     removeBlock(set[i],b[i]))" .
                                                     Validator[i] -> Vote_Manager :
                                                     (["1*1"] " "&&"(Votes'=addVote(
                                                     Votes,b[i],stake[i]))".PoS ))
                                          }
                                  }
                   }
                   else{
                          PoS
                   }
          +["rC*1"] "(lastCheck[i]'=extractCheckpoint(listCheckpoints[i],lastCheck[i
               ]))&(heightLast[i]'=getHeight(extractCheckpoint(listCheckpoints[i],
               lastCheck[i])))&(votes[i]'=calcVotes(Votes,extractCheckpoint(
               listCheckpoints[i],lastCheck[i])))"&&" " .
                   if "(heightLast[i]=heightCheckpoint[i]+EpochSize)&(votes[i]>=2/3*
                        tot_stake)"@Validator[i] then{
                          if "(heightLast[i]=heightCheckpoint[i]+EpochSize)"@Validator[
                               i] then{
                                  ["1"] "(lastJ[i]'=b[i])&(L[i]'= updateHF(L[i],lastJ[i
                                       ]))" @Validator[i].Validator[i]->Vote_Manager
                                       :(["1*1"]" "&&"(epoch'=height(lastF(L[i]))&(Stakes
                                       '=addVote(Votes,b[i],stake[i]))".PoS)
                          }
                          else{
                                  ["1"] "(lastJ[i]'=b[i])"@Validator[i] . PoS
                          }
                   }
                   else{
                          PoS
                   }
         )
}
```

Listing 9 Choreographic language for the Hybrid Casper Protocol.

```
module Validator1
   ...

   [] (Validator1=0) → mR : (b1'=createB(b1,L1,c1))&(c1'=c1+1)&(Validator1'=1);
   [] (Validator1=0) → lR : (Validator1'=2);
   [] (Validator1=0)&(!isEmpty(listCheckpoints1)) →
        rC : (lastCheck1'=extractCheckpoint(listCheckpoints1,lastCheck1))&(
             heightLast1'=getHeight(extractCheckpoint(listCheckpoints1,lastCheck1)))
             &(votes1'=calcVotes(Votes,extractCheckpoint(listCheckpoints1,lastCheck1
             )))&(Validator1'=3);
```

```
   [NGRDF] (Validator1=1)&!(mod(getHeight(b1),EpochSize)=0) → 1 : (L1'=addBlock(
       L1,b1))&(Validator1'=0);
   [] (Validator1=1)&!(!(mod(getHeight(b1),EpochSize)=0)) → 1 : (Validator1'=3);
   [PCRLD] (Validator1=1)&!(mod(getHeight(b1),EpochSize)=0) →
       1 : (L1'=addBlock(L1,b1))&(Validator1'=4);
   [VSJBE] (Validator1=5) → 1 : (Validator1'=0);
   [] (Validator1=2)&!isEmpty(set1) →
       1 : (b1'=extractBlock(set1))&(Validator1'=4);
   [] (Validator1=4)&!canBeInserted(L1,b1) → (Validator1'=0);
   [] (Validator1=4)&!(!canBeInserted(L1,b1)) → 1 : (Validator1'=6);
   [MDDCF] (Validator1=6)&!(mod(getHeight(b1),EpochSize)=0) →
       1 : (setMiner1' = addBlockSet(setMiner1 , b1))&(Validator1'=0);
   [] (Validator1=6)&!(!(mod(getHeight(b1),EpochSize)=0)) → 1 : (Validator1'=8);
   [IQVPA] (Validator1=6)&!(mod(getHeight(b1),EpochSize)=0) →
       1 : (setMiner1' = addBlockSet(setMiner1 , b1))&(Validator1'=9);
   [IFNVZ] (Validator1=10) → 1 : (Validator1'=0);
   [] (Validator1=2)&!(!isEmpty(set1)) → 1 : (Validator1'=0);
   [] (Validator1=3)&(heightLast1=heightCheckpoint1+EpochSize)&(votes1>=2/3*
       tot_stake) → (Validator1'=4);
   [] (Validator1=4)&(heightLast1=heightCheckpoint1+EpochSize) →
       1 : (lastJ1'=b1)&(L1'= updateHF(L1,lastJ1))&(Validator1'=6);
   [EQCYO] (Validator1=6) → 1 : (Validator1'=0);
   [] (Validator1=4)&!((heightLast1=heightCheckpoint1+EpochSize)) →
       1 : (lastJ1'=b1)&(Validator1'=0);
   [] (Validator1=3)&!((heightLast1=heightCheckpoint1+EpochSize)&(votes1>=2/3*
       tot_stake)) → 1 : (Validator1'=0);
endmodule
...
module Network
  Network : [0..1] init 0;
  set1 : list [];
  set2 : list [];
  set3 : list [];
  set4 : list [];
  set5 : list [];

   [NGRDF] (Network=0) →
       1 : (set2'=addBlockSet(set2,b2))&(set3'=addBlockSet(set3,b3))&(set4'=
           addBlockSet(set4,b4))&(set5'=addBlockSet(set5,b5))&(Network'=0);
   [PCRLD] (Network=0) →
       1 : (set2'=addBlockSet(set2,b2))&(set3'=addBlockSet(set3,b3))&(set4'=
           addBlockSet(set4,b4))&(set5'=addBlockSet(set5,b5))&(Network'=0);
   [MDDCF] (Network=0) → 1 : (set1' = removeBlock(set1,b1))&(Network'=0);
   [IQVPA] (Network=0) → 1 : (set1' = removeBlock(set1,b1))&(Network'=0);
   ...
endmodule

module Vote_Manager
  Vote_Manager : [0..1] init 0;
  epoch : [0..10] init 0;
  Votes : hash[];
  tot_stake : [0..120000] init 50;
  stake1 : [0..N] init 10;
  stake2 : [0..N] init 10;
  stake3 : [0..N] init 10;
```

```
    stake4 : [0..N] init 10;
    stake5 : [0..N] init 10;

    [VSJBE] (Vote_Manager=0) →
        1 : (Votes'=addVote(Votes,b1,stake1))&(Vote_Manager'=0);
    ...
endmodule
```

Listing 10 Generated PRISM program for the Hybrid Casper Protocol.

The code is very similar to the one presented in [3], the main difference is the fact that our generated model has more lines of code. This is due to the fact that there are some commands that can be merged, but the compiler is not able to do it automatically. This discrepancy between the two models can be observed also in the simulations, reported in Figure 5. Although the results are similar, PRISM takes 39.016 seconds to run the simulations for the generated model, instead of 22.051 seconds needed for the original model.

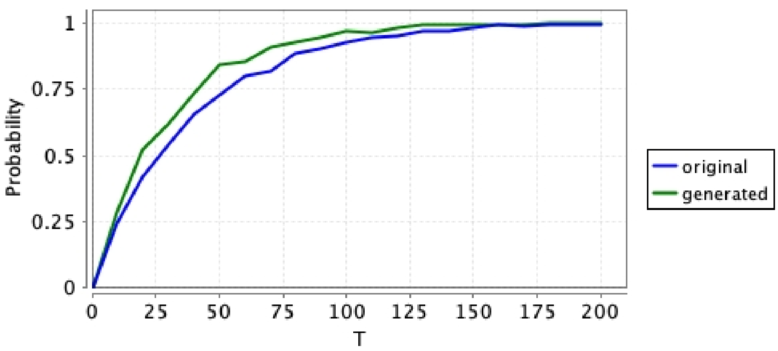


Figure 5 Probability that a block has been created.

2.6 Problems

While testing our choreographic language, we noticed that some of the case studies presented in the PRISM documentation [1] cannot be modeled by using our language. The reasons are various, in this section we try to outline the problems.

- **Asynchronous Leader Election**[4]: processes synchronize with the same label but the conditions are different. We include in our language the `it-then-else` statement but we do not allow the `if-then` (without the `else`). This is done because in this way, we do not incur in deadlock states.
- **Probabilistic Broadcast Protocols**[5]: also in this case, the problem are the labels of the synchronizations. In fact, all the processes synchornize with the same label on every actions. This is not possible in our language, since a label is unique for every synchronization between two (or more) processes.

[4] https://www.prismmodelchecker.org/casestudies/asynchronous_leader.php
[5] https://www.prismmodelchecker.org/casestudies/prob_broadcast.php

- **Cyclic Server Polling System**[6]: in this model, the processes $\texttt{station}_i$ do two different things in the same state. More precicely, at the state 0 ($\texttt{s}_i\texttt{=0}$), the processes may synchornize with the process `server` or may change their state without any synchronization. In out language, this cannot be formalized since the synchronization is a branch action, so there should be another option with a synchronization.

References

1 Prism documentation. https://www.prismmodelchecker.org/. Accessed: 2023-09-05.

2 Stefano Bistarelli, Rocco De Nicola, Letterio Galletta, Cosimo Laneve, Ivan Mercanti, and Adele Veschetti. Stochastic modeling and analysis of the bitcoin protocol in the presence of block communication delays. *Concurr. Comput. Pract. Exp.*, 35(16), 2023. doi:10.1002/cpe.6749.

3 Letterio Galletta, Cosimo Laneve, Ivan Mercanti, and Adele Veschetti. Resilience of hybrid casper under varying values of parameters. *Distributed Ledger Technol. Res. Pract.*, 2(1):5:1–5:25, 2023. doi:10.1145/3571587.

4 D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[6] https://www.prismmodelchecker.org/casestudies/polling.php