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

1 Formal Language

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

1.1 Choreographies

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

$$
\begin{array}{llll}
(\text{Chor}) & C & ::= & \{\mathsf{p}_i\}_{i \in I} + \{\lambda_j : x_j = E_j;\ C_j\}_{j \in J} \ \mid\ \text{if } E@\mathsf{p} \text{ then } C_1 \text{ else } C_2 \ \mid\ X \ \mid\ \mathbf{0} \\
(\text{Expr}) & E & ::= & f(\tilde{E}) \ \mid\ x \ \mid\ v \\
(\text{Rates}) & \lambda \in \mathbb{R} & & (\text{Variables}) \quad x \in \mathsf{Var} \qquad (\text{Values}) \quad v \in \mathsf{Val}
\end{array}
$$

We briefly comment the various constructs. The syntactic category C denotes choreographic programmes. The term $\mathsf{p} \longrightarrow \{\mathsf{p}_i\}_{i \in I} \oplus \{[\lambda_j]x_j = E_j : C_j\}_{j \in J}$ denotes an interaction between roles p_i...

1.2 PRISM

Syntax.

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

Semantics. We construct all the enables commands by applying a closure to the following

24   rules.

$$\frac{[]E \to \{\lambda_i : x_i = E_i\}_{i \in I} \in \{[M_j]\} \quad j \in \{1, 2\}}{[]E \to \{\lambda_i : x_i = E_i\}_{i \in I} \in \{[M_1|[A]|M_2]\}}$$

25

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

$$\frac{[a]E \to \{\lambda_j : x_i = E_i\}_{i \in I} \in \{[M_1]\} \quad [a]E' \to \{\lambda_j : x'_j = E'_j\}_{j \in J} \in \{[M_2]\} \quad a \in A}{[a]E \wedge E' \to \{\lambda_i * \lambda'_j : x_i = E_i \wedge x'_j = E'_j\}_{i \in I, j \in J} \in \{[M_1|[A]|M_2]\}}$$

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

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

28

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

$$\sigma : X \to V$$

1.3 Projection from Choreographies to PRISM

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

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

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

$$=$$

```
label = newlabel();
for p_k ∈ roles{
  for j ∈ J{
    network = add(p_k, [label]s_{p_k} = state(p_k) →  λ_j : x_j = E_j & s'_{p_k} = genNewState(p_k));
  }
}
for j ∈ J{
  network = f(D_j, network);
}
return network
```

$$f\Big(\ \text{if } E@\mathsf{p} \text{ then } C_1 \text{ else } C_2, \mathtt{network}\Big)$$

$$=$$

34

$$\mathtt{network} = \mathtt{add}(\mathsf{p}, [\,]s_\mathsf{p} = \mathtt{state}(\mathsf{p}) \ \& \ f(E));$$
$$\mathtt{network} = f(C_1, \mathtt{network});$$
$$\mathtt{network} = f(C_2, \mathtt{network});$$
$$\mathtt{return} \ \mathtt{network}$$

2 Tests

We tested our language by various examples.

2.1 The Dice Program

The first example we present is the Dice Program[1] [2]. 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.
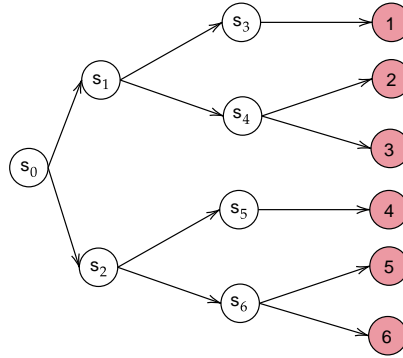
We modelled the program using the choreographic language (Listing 1) and we were able to generate the corresponding PRISM program, reported in Listing 2.

```
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

```
69
70   DiceProtocol₃ ≔ Dice → Dice : (["1*1"] " "&&" ".DiceProtocol₃)
71
72   }
```

■ **Listing 1** Choreographic language for the Dice Program.

```
73
74   dtmc
75
76   module Dice
77          Dice : [0..11] init 0;
78          d : [0..6] init 0;
79
80          []  (Dice=0)  → 0.5 : (Dice'=2) + 0.5 :  (Dice'=6);
81          []  (Dice=2)  → 0.5 : (Dice'=3) + 0.5 :  (Dice'=4);
82          []  (Dice=3)  → 0.5 : (Dice'=2) + 0.5 : (d'=1)&(Dice'=10);
83          []  (Dice=4)  → 0.5 : (d'=2)&(Dice'=10) + 0.5 : (d'=3)&(Dice'=10);
84          []  (Dice=6)  → 0.5 : (Dice'=7) + 0.5 :  (Dice'=8);
85          []  (Dice=7)  → 0.5 : (Dice'=6) + 0.5 : (d'=4)&(Dice'=10);
86          []  (Dice=8)  → 0.5 : (d'=5)&(Dice'=10) + 0.5 : (d'=6)&(Dice'=10);
87          []  (Dice=10) → 1 : (Dice'=10);
88
89   endmodule
90
```

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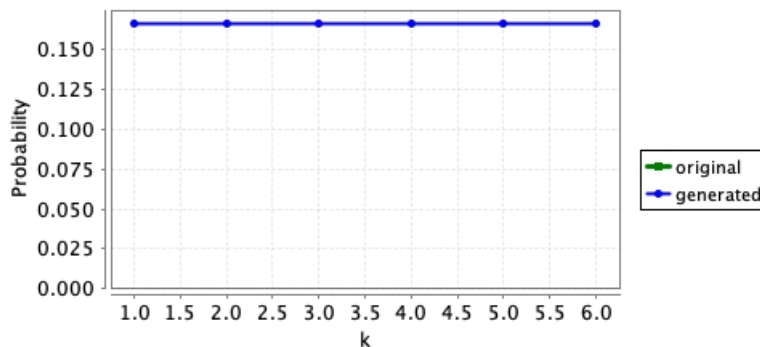
■ **Figure 1** Probability of reaching a state where $d = k$, for $k = 1, \ldots, 6$.

2.2 Simple Peer-To-Peer Protocol

⁹³

⁹⁴ This case study describes a simple peer-to-peer protocol based on BitTorrent[2]. The model
⁹⁵ comprises a set of clients trying to download a file that has been partitioned into K blocks.
⁹⁶ Initially, there is one client that has already obtained all of the blocks and N additional
⁹⁷ clients with no blocks. Each client can download a block from any of the others but they can
⁹⁸ only attempt four concurrent downloads for each block.
⁹⁹ 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.

¹²⁷ Part of the generated PRISM code is shown in Listing 4 and it is faithful with what
¹²⁸ 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];
```

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

```
141          b12 : [0..1];
142          b13 : [0..1];
143          b14 : [0..1];
144          b15 : [0..1];
145
146          [] ( Client1=0) → rate1 : (b11'=1)&(Client1'=0);
147          [] ( Client1=0) → rate2 : (b12'=1)&(Client1'=0);
148          [] ( Client1=0) → rate3 : (b13'=1)&(Client1'=0);
149          [] ( Client1=0) → rate4 : (b14'=1)&(Client1'=0);
150          [] ( Client1=0) → rate5 : (b15'=1)&(Client1'=0);
151
152    endmodule
153
```

■ **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 $0 \le T \le 1.5$ both for our generated model and the model reported in the documentation.
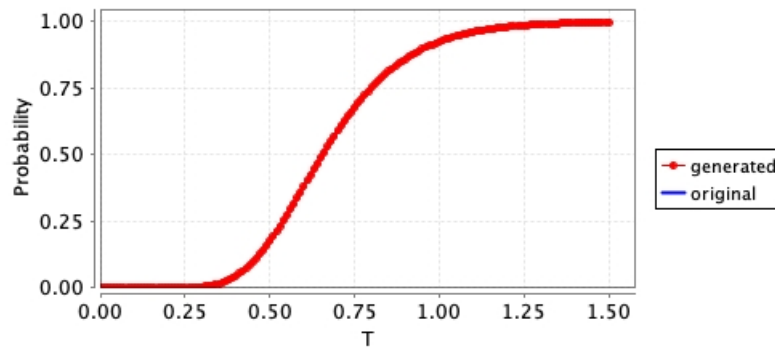
■ **Figure 2** Probability that clients received all the block before T, with $0 \le T \le 1.5$.

2.3 Proof of Work Bitcoin Protocol

This protocol represents the Proof of Work implemented in the Bitcoin blockchain. In[1], 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.

```
165    preamble
166    "ctmc"
167    "const T"
168    "const double r = 1;"
169    "const double mR = 1/600;"
170    "const double lR = 1-mR;"
171    "const double hR1 = 0.25;"
172    "const double hR2 = 0.25;"
173    "const double hR3 = 0.25;"
```

```
174   "const double hR4 = 0.25;"
175   "const double rB = 1/12.6;"
176   "const int N = 100;"
177   endpreamble
178
179   n = 4;
180
181   Hasher[i] -> i in [1...n] ;
182
183   Miner[i] -> i in [1...n]
184   Miner[i] : "b[i] : block {m[i],0;genesis,0} ;", "B[i] : blockchain [{genesis,0;
185       genesis,0}];" ,"c[i] : [0..N] init 0;", "setMiner[i] : list [];" ;
186
187   Network ->
188   Network : "set1 : list [];", "set2 : list [];", "set3 : list [];" , "set4 : list
189       [];";
190
191   {
192   PoW := Hasher[i] → Miner[i] :
193   (+["mR*hR[i]"] " "&&"(b[i]'=createB(b[i],B[i],c[i]))&(c[i]'=c[i]+1)" .
194       Miner[i] → Network :
195           (["rB*1"] "(B[i]'=addBlock(B[i],b[i]))" &&
196           foreach(k != i) "(set[k]'=addBlockSet(set[k],b[i]))" @Network .PoW)
197    +["lR*hR[i]"] " " && " " .
198       if "!isEmpty(set[i])"@Miner[i] then {
199           ["r"] "(b[i]'=extractBlock(set[i]))"@Miner[i] .
200               Miner[i] → Network :
201               (["1*1"] "(setMiner[i]' = addBlockSet(setMiner[i] , b[i]))"
202                   &&"(set[i]' = removeBlock(set[i],b[i]))" . PoW)
203       }
204       else{
205           if "canBeInserted(B[i],b[i])"@Miner[i] then {
206               ["1"] "(B[i]'=addBlock(B[i],b[i]))
207               &(setMiner[i]'=removeBlock(setMiner[i],b[i]))"@Miner[i] . Pow
208           }
209           else{
210               PoW
211           }
212       }
213   )
214   }
215
```

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

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

```
217
218   ctmc
219   const T;
220   const double r = 1;
221   const double mR = 1/600;
222   const double lR = 1−mR;
223   const double hR1 = 0.25;
224   const double hR2 = 0.25;
225   const double hR3 = 0.25;
226   const double hR4 = 0.25;
```

```
227    const double rB = 1/12.6;
228    const int N = 100;
229
230    module Miner1
231    Miner1 : [0..7] init 0;
232    b1 : block {m1,0;genesis,0} ;
233    B1 : blockchain [{ genesis ,0; genesis ,0 }];
234    c1 : [0..N] init 0;
235    setMiner1 : list [];
236
237    [PZKYT] (Miner1=0) → hR1 : (b1'=createB(b1,B1,c1))&(c1'=c1+1)&(Miner1'=1);
238    [EUBVP] (Miner1=0) → hR1 : (Miner1'=2);
239    [HXYKO] (Miner1=1) → 1 : (B1'=addBlock(B1,b1))&(Miner1'=0);
240    []   (Miner1=2)&!isEmpty(set1) → r : (b1'=extractBlock(set1))&(Miner1'=4);
241    [SRKSV] (Miner1=4) → 1 : (setMiner1' = addBlockSet(setMiner1 , b1))&(Miner1'=0);
242    []   (Miner1=2)&!(!isEmpty(set1)) → 1 : (Miner1'=5);
243    []   (Miner1=5)&canBeInserted(B1,b1) → 1 : (B1'=addBlock(B1,b1))
244                    &(setMiner1'=removeBlock(setMiner1,b1))&(Miner1'=0);
245    []   (Miner1=5)&!(canBeInserted(B1,b1)) → 1 : (Miner1'=0);
246    endmodule
247    . . .
248    module Network
249    Network : [0..1] init 0;
250    set1 : list [];
251    . . .
252
253    [HXYKO] (Network=0) → 1 : (set2'=addBlockSet(set2,b2))&(set3'=addBlockSet(set3,
254        b3))&(set4'=addBlockSet(set4,b4))&(Network'=0);
255    [SRKSV] (Network=0) → 1 : (set1' = removeBlock(set1,b1))&(Network'=0);
256    . . .
257
258    endmodule
259
260    module Hasher1
261    Hasher1 : [0..1] init 0;
262
263    [PZKYT] (Hasher1=0) → mR : (Hasher1'=0);
264    [EUBVP] (Hasher1=0) → lR : (Hasher1'=0);
265
266    endmodule
267
```
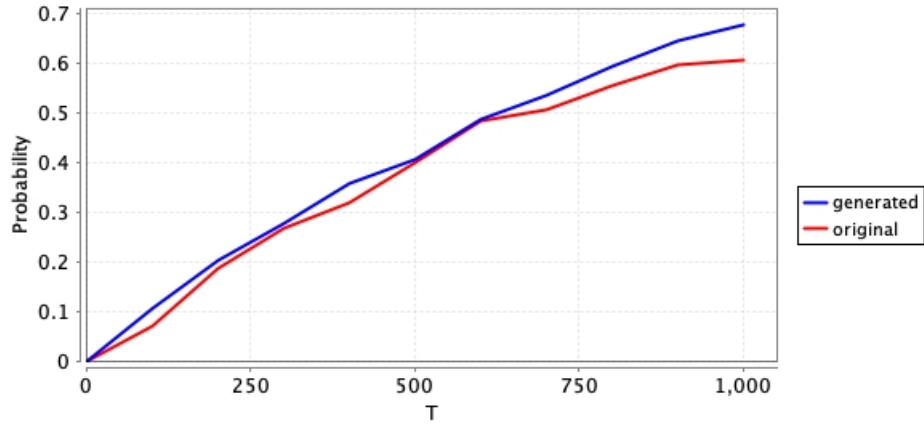
■ **Listing 6** Generated PRISM program for the Peer-To-Peer Protocol.

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 [1].

2.4 Random Graphs Protocol

```
        preamble
```

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

```
273
274        "mdp"
275        "const double p;"
276
277        endpreamble
278
279        n = 3;
280
281        PC ->
282        PC : "s : [1..n];";
283
284
285        Module[i] -> i in [1...n]
286        Module[i] : "varModule[i] : bool;";
287
288        Process[i] -> i in [1...n]
289        Process[i] : "var[i] : bool;";
290
291        {
292        GraphConnected0 :=
293             if "(s=[i])"@PC then {
294                   PC -> Module[i] : (+["1*p"] " "&&"(varModule[i]'=true)". END
295                                                 +["1*(1-p)"] " "&&"(
296                                                         varModule[i]'=true)".
297                                                       END)
298             }
299             if "(s=[i+3])"@PC then {
300                   PC -> Process[i] : (+["1*p"] " "&&"(var[i]'=true)" . END
301                                                 +["1*(1-p)"] " "&&"(var[i]'=false)".
302                                                 if "(PC=6)&!var[i]&(false | (var[i] &
303                                                     varModule[i]) | (varModule[i+1] &
304                                                     var[i+2])) "@Process[i] then {
305                                                         ["1"]"(var[i]'=true)"@Process[i]
306                                                                 . GraphConnected0
307                                                 }
308                             )
```

```
309          }
310        }
311
```

🟧 **Listing 7** Choreographic language for the Random Graphs Protocol.

Part of the generated PRISM code is shown in Listing 8.

```
314   mdp
315   const double p;
316
317   module PC
318   PC : [0..7] init 0;
319   s : [1..7];
320
321   [XFHRA] (PC=0) −> 1 : (PC'=1);
322   [FJFXP] (PC=1) −> 1 : (PC'=2);
323   [HQMAI] (PC=2) −> 1 : (PC'=3);
324   [LSNRS] (PC=3) −> 1 : (PC'=4);
325   [HJHOG] (PC=4) −> 1 : (PC'=5);
326   [IPDQV] (PC=5) −> 1 : (PC'=6);
327
328   endmodule
329
330   module Process1
331   Process1 : [0..3] init 0;
332   var1 : bool;
333
334   [LSNRS] true −> p : (var1'=true)+(1−p) : (var1'=false);
335
336   []  (PC=6)& !var1&(false | (var1 & varModule1) | (varModule2 & var3)) −> 1 : (
337       var1'=true);
338   endmodule
339
340   module Process2
341   Process2 : [0..3] init 0;
342   var2 : bool;
343
344   [HJHOG] true −> p : (var2'=true)+(1−p) : (var2'=false);
345
346   []  (PC=6)&!var2&(false | (var2 & varModule2) | (varModule3 & var1)) −> 1 : (
347       var2'=true);
348   endmodule
349
350   module Process3
351   Process3 : [0..3] init 0;
352   var3 : bool;
353
354   [IPDQV]true −> p : (var3'=true)+(1−p) : (var3'=false);
355
```

```
356      []  (PC=6)&!var3&(false | (var3 & varModule3) | (varModule1 & var2)) −> 1 : (
357          var3'=true);
358      endmodule
359
360      module Module1
361      Module1 : [0..1] init 0;
362      varModule1 : bool;
363
364      [XFHRA] (Module1=0) −> p : (varModule1'=true)&(Module1'=0)+(1−p) : (
365          varModule1'=true)&(Module1'=0);
366
367      endmodule
368
369
370      module Module2
371      Module2 : [0..1] init 0;
372      varModule2 : bool;
373
374      [FJFXP] (Module2=0) −> p : (varModule2'=true)&(Module2'=0)+(1−p) : (
375          varModule2'=true)&(Module2'=0);
376
377      endmodule
378
379
380      module Module3
381      Module3 : [0..1] init 0;
382      varModule3 : bool;
383
384      [HQMAI] (Module3=0) −> p : (varModule3'=true)&(Module3'=0)+(1−p) : (
385          varModule3'=true)&(Module3'=0);
386
387      endmodule
388
```

Listing 8 Generated PRISM program for the Random Graphs Protocol.

In Figure 4, we compare the results obtained with the two models.
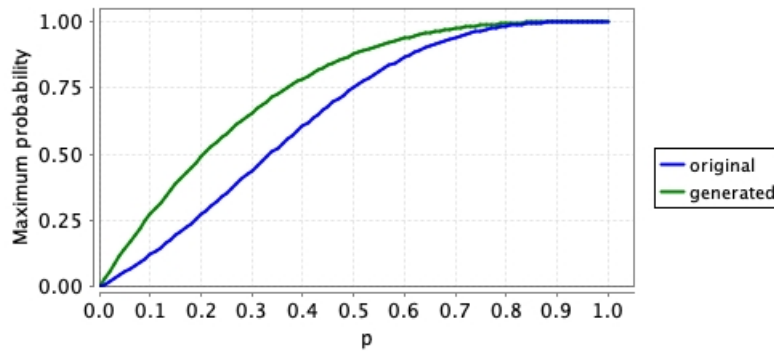
Figure 4 Probability that the nodes 1 and 2 are connected.

References

1 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`.

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