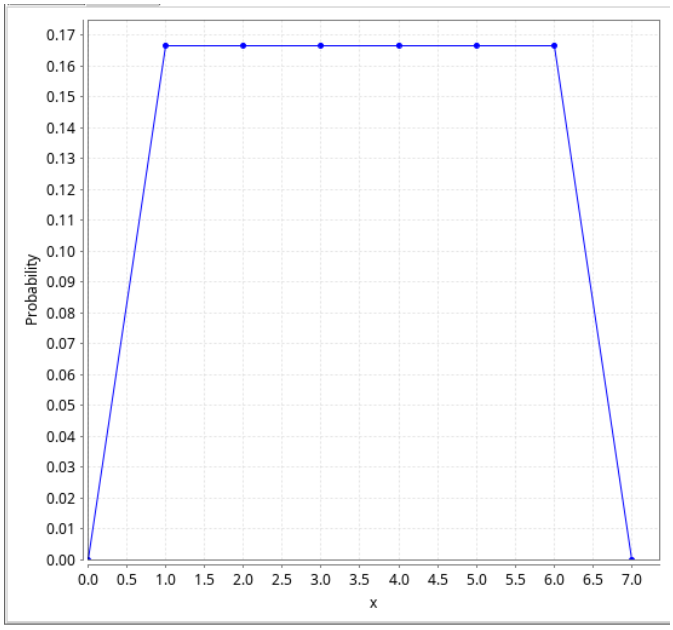

Figure 1: The possibility of having the property P=? [F phi] for $0 \leq x \leq 7$

Part 1

1. Exploring the model in PRISM

   Look at the trace displayed in the main part of the window. What is the final value of s? It should be 7. The final result of s is indeed 7. The value of the die happens to be 2 in this case.

   Try to reach a state where the value of the die is 6. I got a 6 after running the simulation one more time. What a coincidence.

   What is the minimum path length you observe? What is the maximum? When looking at the graphical illustration we see that the minimum path length is 4. We also see that the graph contains loops. This means that the maximum path length is unbounded. I ran the simulation 20 times and the shortest path I came across had length 4 and the longest path had length 10.

2. Model checking with PRISM

   What does phi mean in this case? What does the property mean? In this case we have *phi* equal to

   ```
   F s=7 & d=x
   ```

   meaning that at some point s should be 7, meaning that a die has reached and that d=x, meaning that the die should be on its x-th side.

   Right click on the property in the GUI and select "Verify". Pick a value of x between 1 and 6 and select I picked x=5. The answer I get is 1.6666650772094727. Of the course the answer should have been 1/6. I didn't know that PRISM uses a numerical method, so I was expecting an exact answer. Later I found out that PRISM uses a numerical method with $\epsilon = 10^{-6}$. As we can see the first six digits of the result are the same as the first six digits of 1/6, hence the answer is correct.

   Re-verify the property and see how the answer changes. If I re-verify the property I get the same result. This shows us that the numerical algorithm is determenistic. If I fill in a different value for x. For example 6 or 1 I also get the same result. This shows us that the method to obtain the probability is the same for each die.

   Use PRISM to plot a graph of the value of the property P=? [F phi for values of x between 0 and 7.]

3. Statistical model checking with PRISM

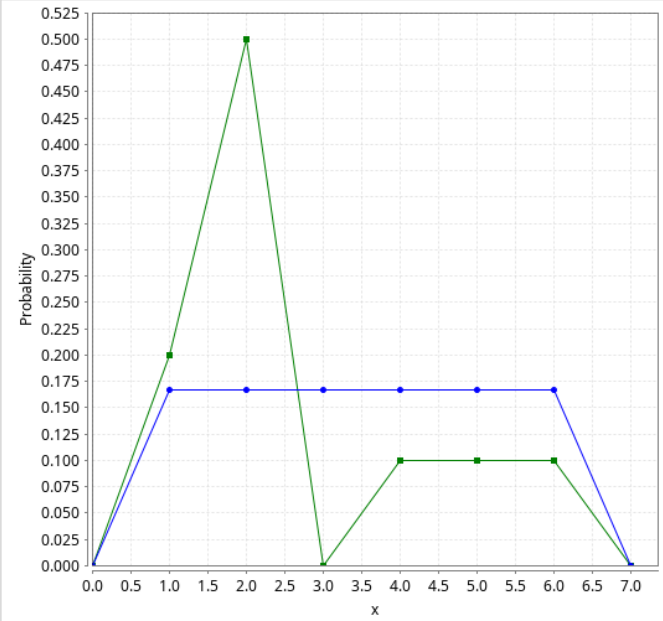

Figure 2: Comparing sample size 10 whith the original

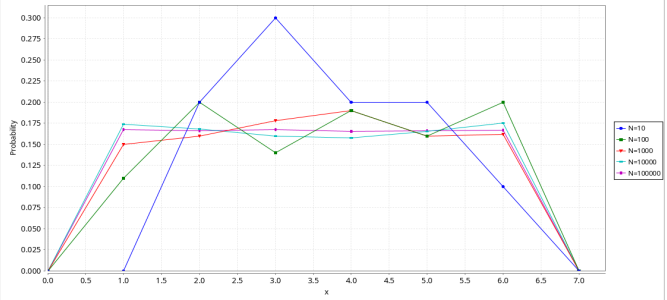

Figure 3: Comparing different sample sizes.

Change the number of samples to 10. How are good are these approximate results? As we can see in figure 2, the chances are a lot less accurate. This is because the law of the large number tells us that the bigger the sample size, the closer the average will be to the expected value. In the case of the bernouli experiment, the expected value is equal to the probability.

How many samples do you need to get results close to those generated through verification? Looking at figure 3, I would say that a sample size of 10000 is a minimum and a 100000 is good.

4. Expected termination time

Compute the expected number of steps that the algorithm being modelled requires to generate a value fo

Let x_s be the expected number of steps in state s. We now that state 7 is always the final state, so $x_7 = 0$. We also see that state 4 and state 5 always go to 7, hence $x_4 = x_5 = 1$. We see that state 3 and 6 will go back to the previous state halve of the time. Hence $x_3 = 1+0.5\cdot x_7+0.5\cdot x_1 = 1+0.5\cdot x_1$ and $x_6 = 1+0.5\cdot x_7+0.5\cdot x_2 = 1+0.5\cdot x_2$. We see that $x_1 = 1+0.5\cdot x_3+0.5\cdot x_4 = 1+0.5\cdot(1+0.5\cdot x_1)+0.5\cdot 1 = 2+0.25x_1$ and $x_2 = 1+0.5\cdot x_5+0.5\cdot x_6 = 1+0.5\cdot 1+0.5\cdot(1+0.5\cdot x_2) = 2+0.25\cdot x_2$, hence $0.75\cdot x_1 = 0.75\cdot x_2 = 2$, thus $x_1 = x_2 = \frac{8}{3}$. Finally, we see that $x_0 = 1+0.5\cdot x_1+0.5\cdot x_2 = \frac{11}{3}$. Our final answer is therefore $\frac{14}{3}$.

What should ϕ be? We want to measure the amount of steps untill we get a die. This is the same as measuring the amount of steps until s=7, hence $\phi = s = 7$.

What is the result? The result is 3.6666650772094727. We see that this is between $\frac{14}{3} - \epsilon$ and $\frac{14}{3} + \epsilon$ and is therefore correct.

Part 5

1. The EGL contract contract signing protocol.

   Look at the first 4 variables in the table representing the path. In the first eight steps, the party constantly switches between 1 and 2, the phase is equal to 1, n inceases every two steps and b is constantly equal to 1. This makes sense as we are looping for(i=1,...,n) and doing OT(A,B,a_i,a_N+i), OT(B,A,b_i,b_N+i). Therefore party is switch between 1 and 2, because it is switching between sending a and sending b. Also n is increasing for the same reason as i is increasing and b is constant, because we are not looking at individual bits at the moment. For the remaining part of the program we see that b increases halfway as the outer loop is looping over the bits. Within the bit loop it switches from A sending to B sending so party switches three times. When n reaches 3 we want to start looking at $n+1, \ldots, 2N$, hence phase switches 7 times. It loops over al the secrets, so n counts from 0 to 3 constantly.

2. Analysing a weakness of the algorithm

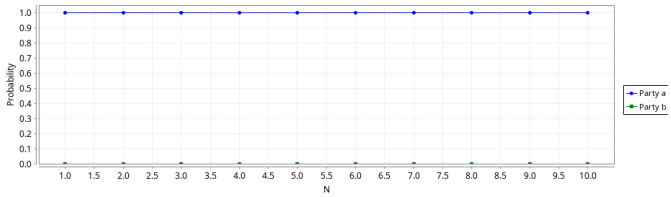

Figure 4:

   Plot the results of the two properties

3. Improving the algorithm: EGL2

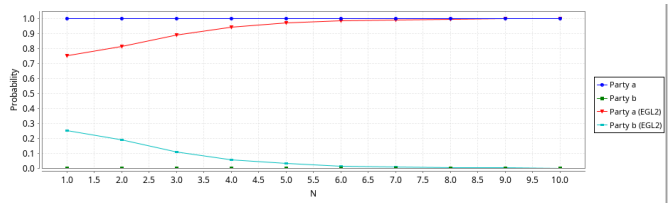

Figure 5:

   Plot the results of the two properties for EGL2

4. Two more versions: EGL3 and EGL4

   EGL3 I replaced the SECOND AND THIRD PHASES with the following code

```
[receiveB] phase=2 & party=1 -> (party'=2);
[receiveA] phase=2 & party=2 & n<N-1 -> (party'=1) & (n'=n+1);
[receiveA] phase=2 & party=2 & n=N-1 -> (phase'=3) & (n'=0) & (party'=1);

[receiveB] phase=3 & party=1 -> (party'=2);
[receiveA] phase=3 & party=2 & n<N-1 -> (party'=1) & (n'=n+1);
[receiveA] phase=3 & party=2 & n=N-1 & b<L -> (phase'=2) & (n'=0) & (party'=1) & (b'=b+1);
[receiveA] phase=3 & party=2 & n=N-1 & b=L -> (phase'=4);
```

   EGL4 Again I only had to replace the SECOND AND THIRD PHASES. This time with the following code

```
[receiveB] phase=2 & party=1 & n=0 -> (party'=2);
[receiveA] phase=2 & party=2 & n<N-1 -> (n'=n+1);
[receiveA] phase=2 & party=2 & n=N-1 -> (n'=1) & (party'=1);
```

```
[receiveB] phase=2 & party=1 & n>0 & n<N-1 -> (n'=n+1);
[receiveB] phase=2 & party=1 & n=N-1 & b<L -> (n'=0) & (b'=b+1);
[receiveB] phase=2 & party=1 & n=N-1 & b=L -> (n'=0) & (b'=1) & (phase'=3);
[] phase=2 & party=1 & n=N & b<L -> (n'=0) & (b'=b+1);
[] phase=2 & party=1 & n=N & b=L -> (n'=0) & (b'=1) & (phase'=3);

[receiveB] phase=3 & party=1 & n=0 -> (party'=2);
[receiveA] phase=3 & party=2 & n<N-1 -> (n'=n+1);
[receiveA] phase=3 & party=2 & n=N-1 -> (n'=1) & (party'=1);
[receiveB] phase=3 & party=1 & n>0 & n<N-1 -> (n'=n+1);
[receiveB] phase=3 & party=1 & n=N-1 & b<L -> (n'=0) & (b'=b+1);
[receiveB] phase=3 & party=1 & n=N-1 & b=L -> (phase'=4);
[] phase=3 & party=1 & n=N & b<L -> (n'=0) & (b'=b+1);
[] phase=3 & party=1 & n=N & b=L -> (phase'=4);
```

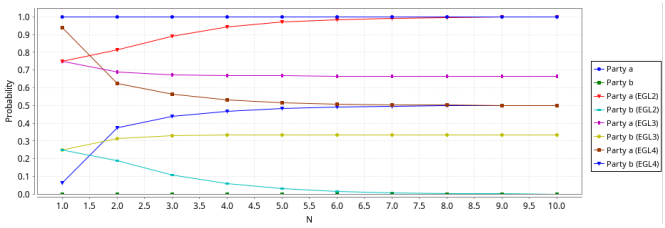

Figure 6:

Graph Other properties of the algorithm(s)

Reward structure I added this code to all the files

```
rewards "messages_A_needs"
 [receiveA] kB & !kA : 1;
endrewards
rewards "messages_B_needs"
 [receiveB] !kB & kA : 1;
endrewards
 \end{description}
```

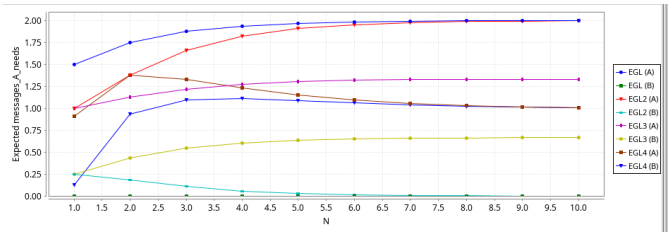

Figure 7: