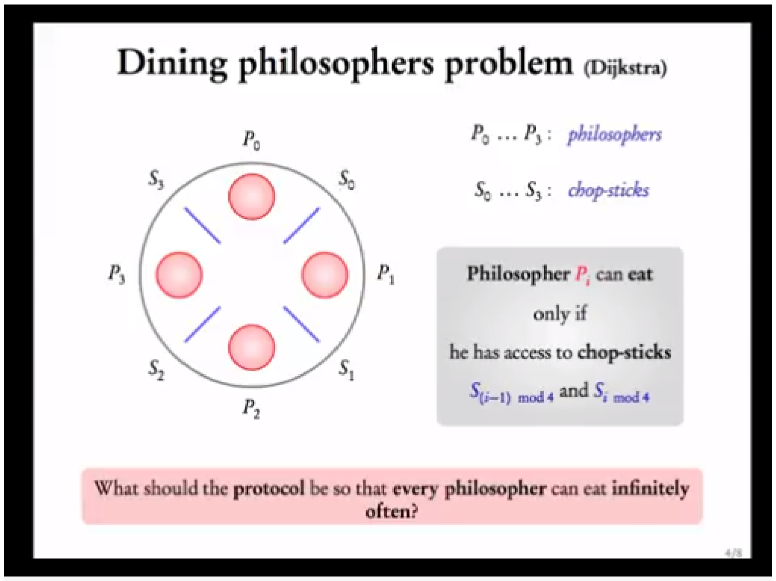
Model Checking

Prof. B. Srivathsan
Department of Computer Science and Engineering
Indian Institute of Technology – Madras

Lecture - 12
A Problem in Concurrency

Welcome to unit 3 of this course. In the first unit we saw how to model controllers as transition systems. The second unit was an instruction to the model checker NuSMV, we also did some examples and exercises. In this unit we will look at more properties that can be checked on models. Let us start this module with a problem in concurrency.

(Refer Slide Time: 00:34)



Dining philosophers problem (Dijkstra)

$P_0 \dots P_3$: *philosophers*

$S_0 \dots S_3$: *chop-sticks*

Philosopher P_i can eat only if he has access to chop-sticks $S_{(i-1) \bmod 4}$ and $S_{i \bmod 4}$

What should the protocol be so that every philosopher can eat infinitely often?

Let P0, P1, P2 and P3 be processes, S0, S1, S2 and S3 be some shared resources. S0 is shared between P0 and P1, S1 is shared between P2 and P1, S2 is shared between P3 and P2 and finally S3 is shared between P0 and P3. A process can execute only if it has access to resources on both sides, for example P1 can execute only if it has access to resources S0 and S1.

Here if I is 1 you would get that it has access to resources S0 and S1. Similarly, P2 can execute only if it has access to S1 and S2, P3 can execute only if it has access to S2 and S3. For P0 it can execute only if it has access to S0 and S3, for P0 Si is S0, but Si minus 1 is P minus 1.

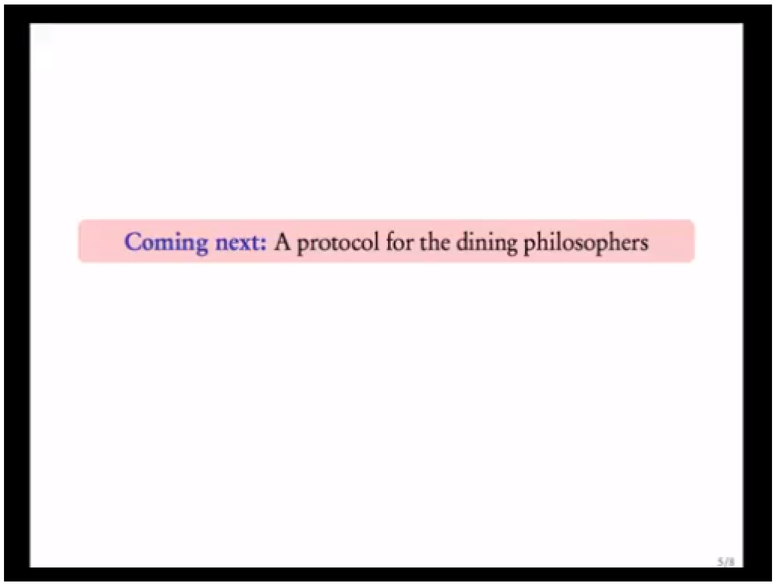
However we have to count modulo 4, in the sense, minus 1 is 3 modulo 4, this is a just a way to say that p0 can execute only if it has access to S0 and S3. The problem now is that there is an operating system which schedules these processes based on the availability of resources. The question is how should the processes be scheduled so that every process gets to execute infinitely often.

This is a canonical problem in operating systems, a different version of this problem is popularly known as the Dining philosophers problem, it was proposed by Dijkstra. Here P0, P1, P2 and P3 are philosophers; S0, S1, S2, S3 are chop-sticks. Assume that these are plates, each philosophers work is to think and to eat, to think and to eat, and so on.

To eat he needs to have these 2 chop sticks, he cannot eat with 1 or none and he is not allowed to eat with hands. So, for philosopher P1 he can eat only if he has access to chop-sticks S0 and S1, similarly P0 can eat only if he has access to S0 and S3 and so on. Just that instead of processes we call them philosophers, instead of shared resources we have shared chop sticks.

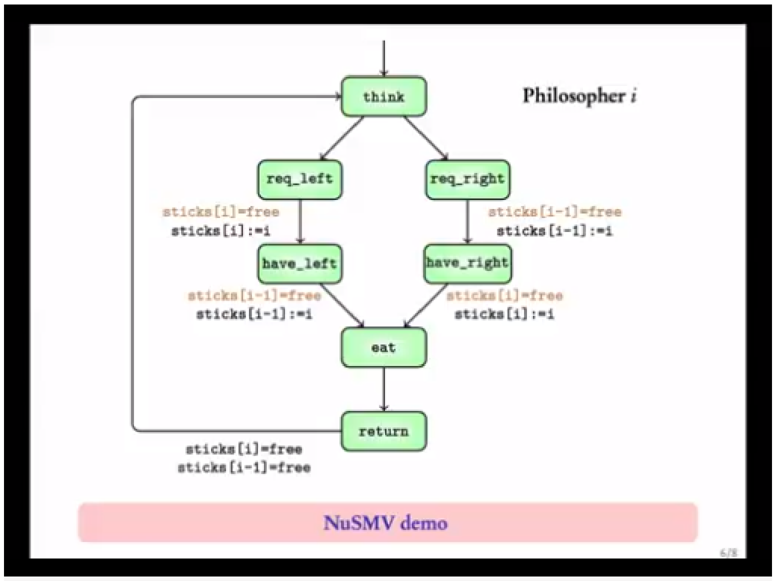
The question is what should the protocol be so that every philosopher can eat infinitely often. By protocol what is the rule that these philosophers should follow in order to access the resources so that each one gets to eat infinitely often. This is a very famous problem.

(Refer Slide Time: 04:28)


Coming next: A protocol for the dining philosophers

Let us now give a protocol for the dining philosophers. This is along the same lines as the Mutual exclusion problem. We will try solve to this problem using our model of transition systems.

(Refer Slide Time: 04:46)



NuSMV demo

Let us try to give a transition system for philosopher I. Initially, the philosopher I is thinking, then he has 2 choices either he requests for the left stick or he request for the right stick. Suppose he requests for the left stick, if the left stick is free.

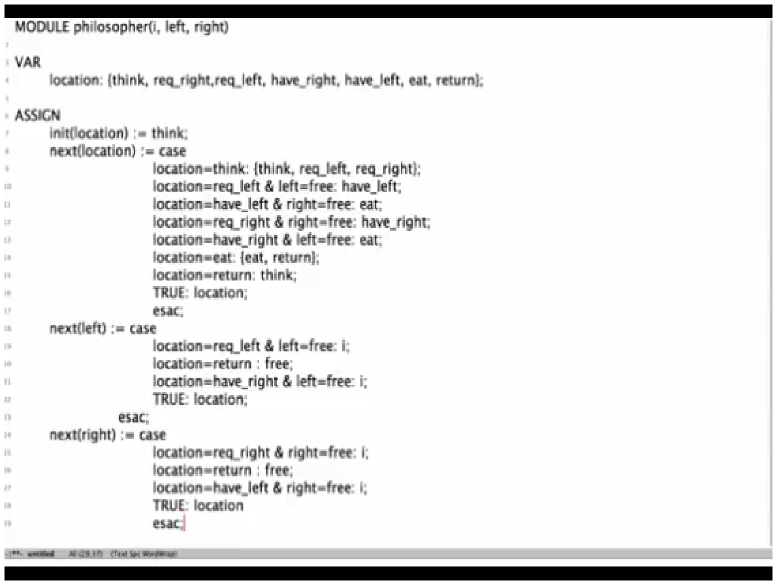
Now, for philosopher P1 his left is S1 assumed that he is sitting facing the plate his left would be S1, his right would be S0, so for philosopher Pi his left is Si, his right is Si minus 1.

So, if his left stick is free then he take the transition to the state have left, in the process he sets that the stick I is with him. Okay, so assuming that sticks is an array, sticks of is being to set to I. Suppose he has left, now he has to wait for the right stick, if the right stick is free, he can take this transition and in the process he denotes that the I minus 1 stick is with him and then he can go to the state where he can eat.

This path is symmetric, here he first requests for the right then he gets his right, then he requests for his left once he has both of them he can eat. He can keep eating as long as he wants and then when he is done he goes to the return state. From the return state he can go back to think, in the process he can release both the sticks. He can say that my resources are free now, my chop sticks are free for you to use. This is the model of the philosopher.

Let us now try to see the NuSMV demo of this model. Let us now write the philosopher module in NuSMV.

(Refer Slide Time: 07:40)



```
MODULE philosopher(i, left, right)

VAR
        location: {think, req_right,req_left, have_right, have_left, eat, return};

ASSIGN
        init(location) := think;
        next(location) := case
                              location=think: {think, req_left, req_right};
                              location=req_left & left=free: have_left;
                              location=have_left & right=free: eat;
                              location=req_right & right=free: have_right;
                              location=have_right & left=free: eat;
                              location=eat: {eat, return};
                              location=return: think;
                              TRUE: location;
                              esac;
        next(left) := case
                              location=req_left & left=free: i;
                              location=return : free;
                              location=have_right & left=free: i;
                              TRUE: location;
                        esac;
        next(right) := case
                              location=req_right & right=free: i;
                              location=return : free;
                              location=have_left & right=free: i;
                              TRUE: location
                              esac;
```

So, we define a module philosopher which takes as input 3 parameters its number, the left stick and the right stick. What are its variables? We need to define its locations the locations are think, request right, request left, have right, have left, eat, return yes that's it. What are the transitions?

The initial value of location is think let us now define the transitions. Next of location is as follows; if location is think, he can either keep thinking or he can request for the left stick or he can request for the right stick. If location is request left and the left stick is free, assume that this gives us the fact that the left stick is free, we will then call it appropriately in the main module.

So, if location is request left and the left stick is free, the philosopher goes to have left. Now, if the location is have left and if the right is free, then he can go to eat. Similarly, if the location is the request right and the right is free, he goes to the state have right and if location is have right and the left is free he goes to the eat state.

When he is in the eat state he can continue eating or he can go to the return state and once he is in the return state, he goes back to the think state. Now, we need to define the next for the variables left and right. Next of left if you see, this is the left stick if the location is request left, then the left stick should be set to the value I and when going back to think the left stick should be made free again.

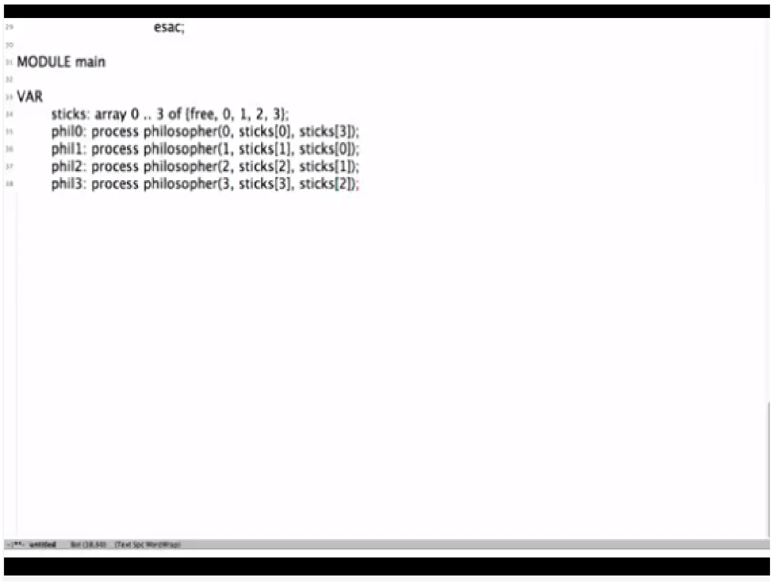
Let us now write it. If location is request left and the left is free, my location goes to have left and my left should be set to I. Now, if location is returned the left should be set to free. This will become clear when we write the main module. Now, what about next of right; if location is request right and the right is free, then right should become I, the same time if location is returned then the right should become free.

We still have missed something, in this part if you are in have right and the left is free then you go to eat. So, in this transition again the left should be set to I, similarly here the right should be set to I.

Lets come back here, if location is have right and left is free, then set your left to I. Similarly, if the location is have left and left is free sorry, right is free and set the right to be I. We are more or less done, however we need to give the statement which says that if none of the conditions match stay in the same location.

Finally, location we have written the philosopher module now let us write the main module.

(Refer Slide Time: 14:40)



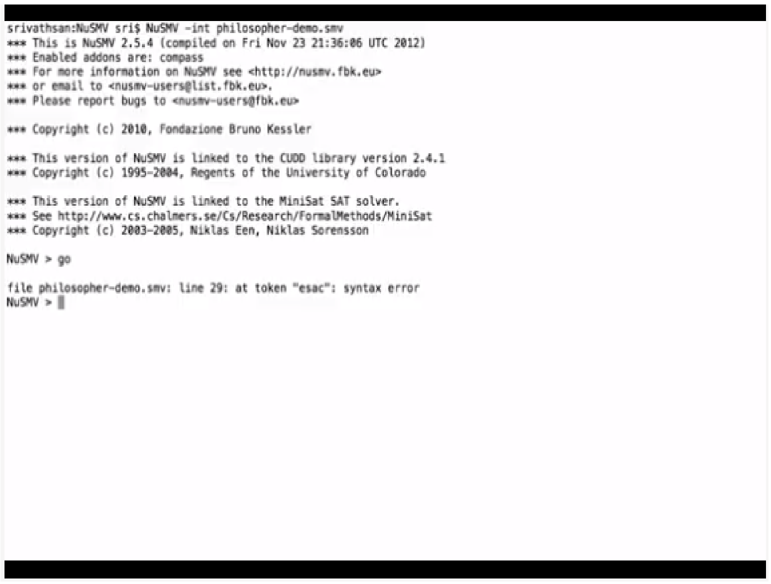
```
                        esac;

MODULE main

VAR
        sticks: array 0 .. 3 of {free, 0, 1, 2, 3};
        phil0: process philosopher(0, sticks[0], sticks[3]);
        phil1: process philosopher(1, sticks[1], sticks[0]);
        phil2: process philosopher(2, sticks[2], sticks[1]);
        phil3: process philosopher(3, sticks[3], sticks[2]);
```

Module main, what are the variables? we will now define a variable denoting sticks it is going to be an array of size the number of philosophers. Let us say the number of philosophers is 4 as given in the slides, 0 to 3 each cell of the array is made of values from this set either it can be free or it can be 0, 1, 2, 3, denoting the fact that philosopher 0 has access to that stick.

Essentially sticks of 0, sticks of 1, sticks of 2, sticks of 3 each of them contain either free 0,1,2 or 3. Lets now define the philosophers; philosopher 0 is a process philosopher his number is 0 if you remember we have 3 inputs to philosopher. His left stick is sticks of 0 and his right stick is sticks of 3. Similarly, philosopher 1 is process, philosopher his number is 1 his left is 1, his right is 0.

Let me complete philosopher 2 his number is 2, his left is 2 and his right is 1. Finally, philosopher 3 process philosopher 3, his left is 3 and his right is 2. We will now run this code in NuSMV.

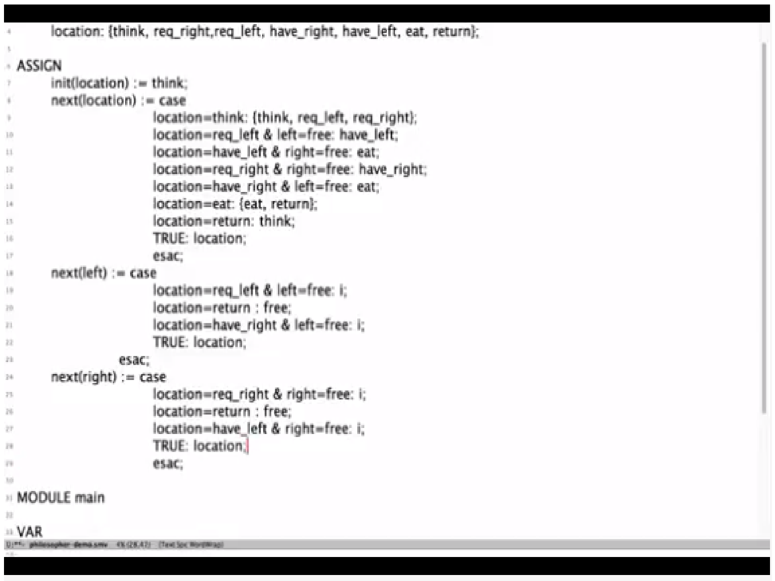
(Refer Slide Time: 16:00)



```
srivathsan:NuSMV sri$ NuSMV -int philosopher-demo.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go

file philosopher-demo.smv: line 29: at token "esac": syntax error
NuSMV >
```

Let us now run this code. I have saved it under philosopher demo, let me call it using NuSMV demo dot smv, now go. There seems to be a syntax error, let us check it says at line 29 there is a syntax error. Line 29, Yes I forgotten the semicolon here and save it now let me run it again NuSMV philosopher demo.

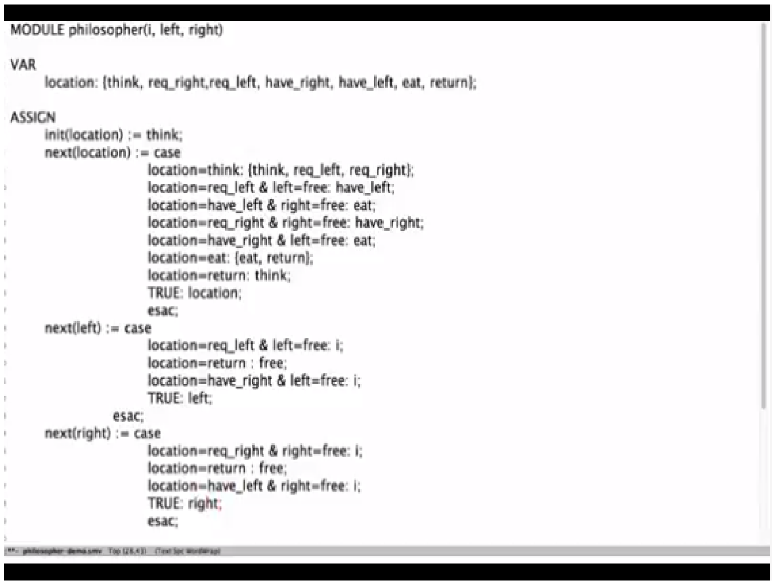
(Refer Slide Time: 16:14)



```
4          location: {think, req_right,req_left, have_right, have_left, eat, return};
5
6   ASSIGN
7          init(location) := think;
8          next(location) := case
9                         location=think: {think, req_left, req_right};
10                        location=req_left & left=free: have_left;
11                        location=have_left & right=free: eat;
12                        location=req_right & right=free: have_right;
13                        location=have_right & left=free: eat;
14                        location=eat: {eat, return};
15                        location=return: think;
16                        TRUE: location;
17                        esac;
18         next(left) := case
19                        location=req_left & left=free: i;
20                        location=return : free;
21                        location=have_right & left=free: i;
22                        TRUE: location;
23                  esac;
24         next(right) := case
25                        location=req_right & right=free: i;
26                        location=return : free;
27                        location=have_left & right=free: i;
28                        TRUE: location;
29                        esac;
30
31  MODULE main
32
33  VAR
U:**-  philosopher-demo.smv    4% (28,47)   (Text Spc WordWrap)
```

There seems to be another problem cannot assign value have right to variable sticks of 3 line 18. So the error is here this should be left and here it should be right so this was the error.

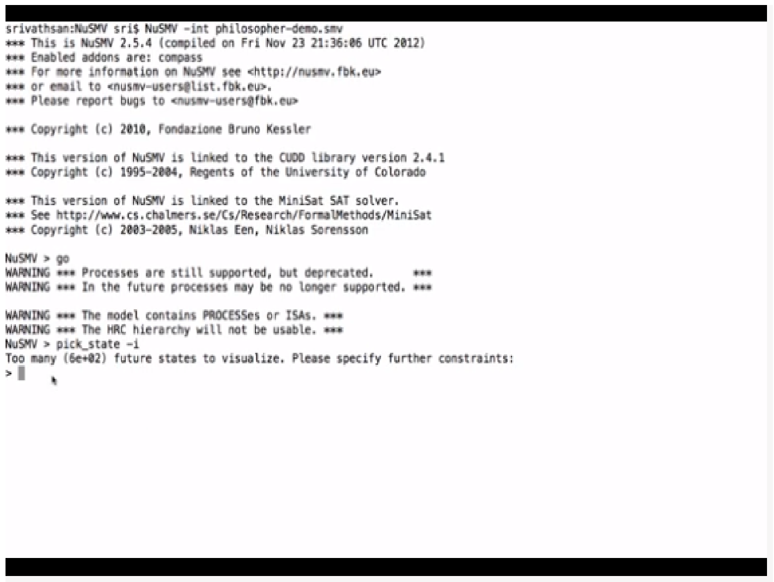
(Refer Slide Time: 16:47)



```
MODULE philosopher(i, left, right)

VAR
        location: {think, req_right,req_left, have_right, have_left, eat, return};

ASSIGN
        init(location) := think;
        next(location) := case
                              location=think: {think, req_left, req_right};
                              location=req_left & left=free: have_left;
                              location=have_left & right=free: eat;
                              location=req_right & right=free: have_right;
                              location=have_right & left=free: eat;
                              location=eat: {eat, return};
                              location=return: think;
                              TRUE: location;
                              esac;
        next(left) := case
                              location=req_left & left=free: i;
                              location=return : free;
                              location=have_right & left=free: i;
                              TRUE: left;
                         esac;
        next(right) := case
                              location=req_right & right=free: i;
                              location=return : free;
                              location=have_left & right=free: i;
                              TRUE: right;
                              esac;
```

Lets run it again, NuSMV philosopher demo, yes it has successfully executed. Let us now try to simulate the module, what is the initial state? if you see there seems to be some error. It says too many future states, this is because we have not specified the initial states in the main module.

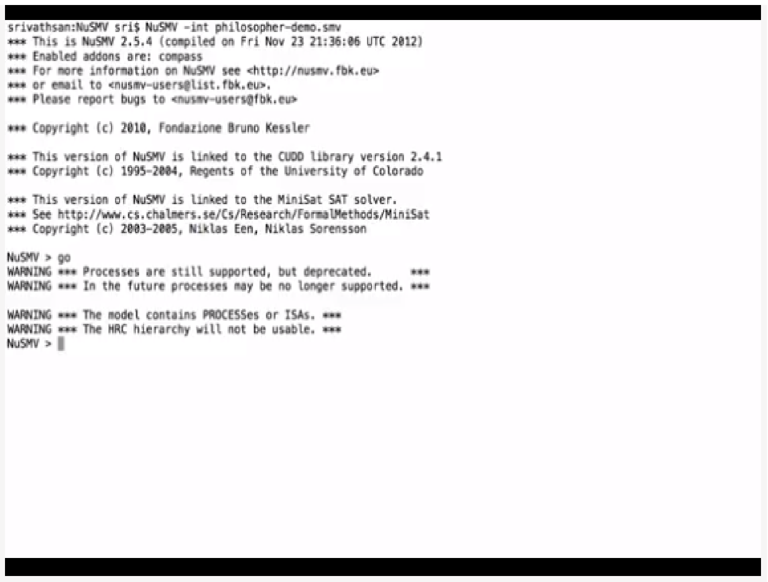
(Refer Slide Time: 17:09)



```
srivathsan:NuSMV sri$ NuSMV -int philosopher-demo.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > pick_state -i
Too many (6e+02) future states to visualize. Please specify further constraints:
>
```

So, in the main module we should have given the initial values for assign initial value of sticks of 0 is free, initial value of sticks of 1 is free as well, initial value of sticks of 2 is free and finally init of sticks of 3 is free as well.

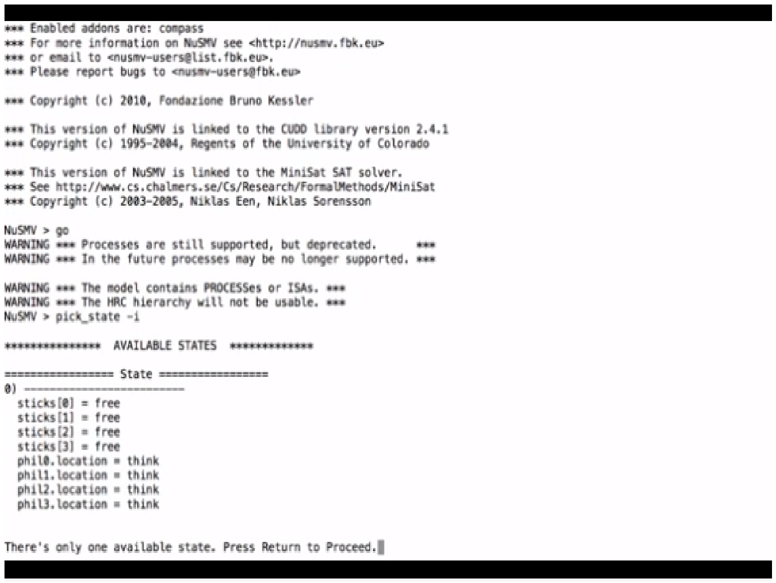
(Refer Slide Time: 18:13)



```
srivathsan:NuSMV sri$ NuSMV -int philosopher-demo.smv
*** This is NuSMV 2.5.4 (compiled on Fri Nov 23 21:36:06 UTC 2012)
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV >
```

Lets hope that we can run it successfully this time NuSMV philosopher. Lets now try to pick an initial state to see if it works, yes it does.

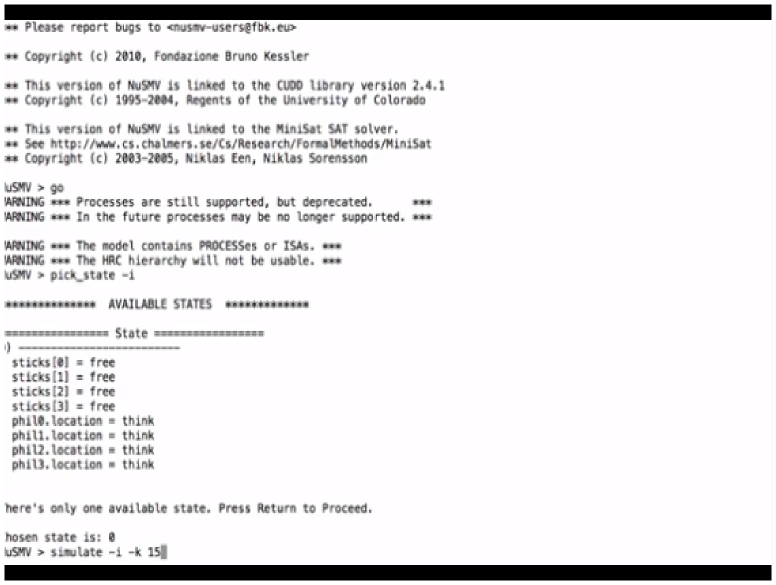
(Refer Slide Time: 18:22)



```
*** Enabled addons are: compass
*** For more information on NuSMV see <http://nusmv.fbk.eu>
*** or email to <nusmv-users@list.fbk.eu>.
*** Please report bugs to <nusmv-users@fbk.eu>

*** Copyright (c) 2010, Fondazione Bruno Kessler

*** This version of NuSMV is linked to the CUDD library version 2.4.1
*** Copyright (c) 1995-2004, Regents of the University of Colorado

*** This version of NuSMV is linked to the MiniSat SAT solver.
*** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
*** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

NuSMV > go
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = think
  phil2.location = think
  phil3.location = think


There's only one available state. Press Return to Proceed.
```

There is only 1 initial state and in this initial state all the philosophers are thinking and all the sticks are free. Let us now try to simulate this model to understand certain things.

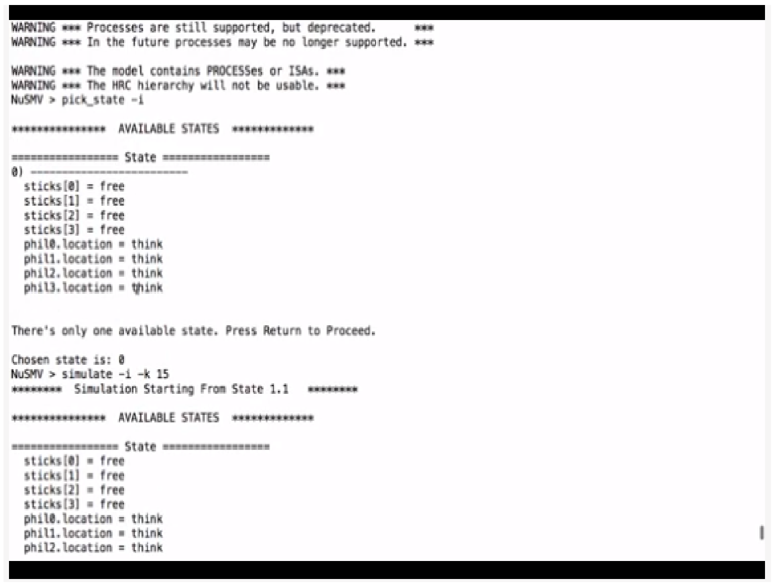
(Refer Slide Time: 18:47)



```
** Please report bugs to <nusmv-users@fbk.eu>

** Copyright (c) 2010, Fondazione Bruno Kessler

** This version of NuSMV is linked to the CUDD library version 2.4.1
** Copyright (c) 1995-2004, Regents of the University of Colorado

** This version of NuSMV is linked to the MiniSat SAT solver.
** See http://www.cs.chalmers.se/Cs/Research/FormalMethods/MiniSat
** Copyright (c) 2003-2005, Niklas Een, Niklas Sorensson

uSMV > go
ARNING *** Processes are still supported, but deprecated.      ***
ARNING *** In the future processes may be no longer supported. ***

ARNING *** The model contains PROCESSes or ISAs. ***
ARNING *** The HRC hierarchy will not be usable. ***
uSMV > pick_state -i

**************  AVAILABLE STATES  *************

================= State =================
) -------------------------
 sticks[0] = free
 sticks[1] = free
 sticks[2] = free
 sticks[3] = free
 phil0.location = think
 phil1.location = think
 phil2.location = think
 phil3.location = think


here's only one available state. Press Return to Proceed.

hosen state is: 0
uSMV > simulate -i -k 15
```

Simulate minus I minus k say for 15 steps.

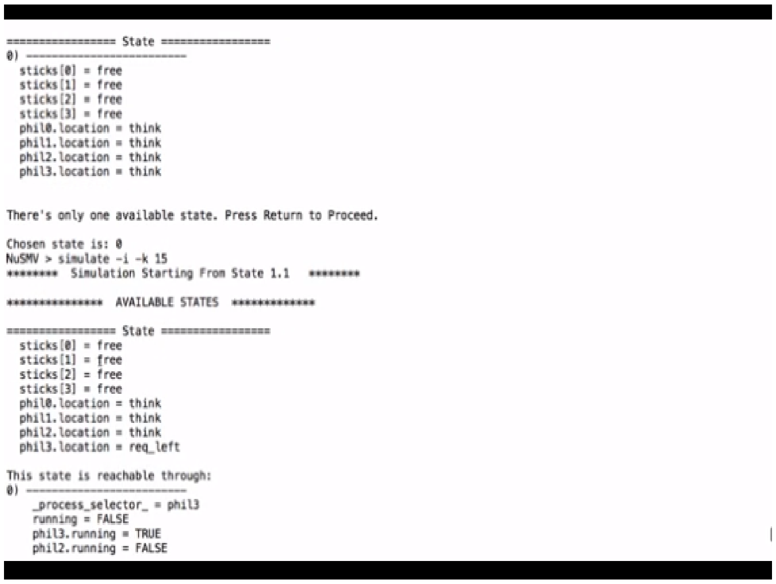
(Refer Slide Time: 18:55)



```
WARNING *** Processes are still supported, but deprecated.      ***
WARNING *** In the future processes may be no longer supported. ***

WARNING *** The model contains PROCESSes or ISAs. ***
WARNING *** The HRC hierarchy will not be usable. ***
NuSMV > pick_state -i

***************  AVAILABLE STATES  *************

================= State =================
0) -------------------------
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = think
  phil2.location = think
  phil3.location = think


There's only one available state. Press Return to Proceed.

Chosen state is: 0
NuSMV > simulate -i -k 15
********  Simulation Starting From State 1.1   ********

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = think
  phil2.location = think
```

We started with all of them in the think state and all sticks free.

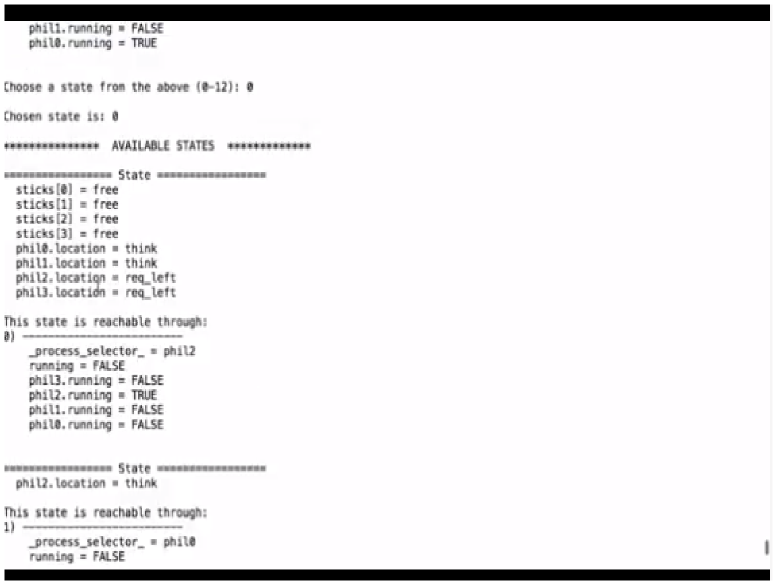
(Refer Slide Time: 19:05)



```
================= State =================
0) -------------------------
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = think
  phil2.location = think
  phil3.location = think


There's only one available state. Press Return to Proceed.

Chosen state is: 0
NuSMV > simulate -i -k 15
********  Simulation Starting From State 1.1   ********

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = think
  phil2.location = think
  phil3.location = req_left

This state is reachable through:
0) -------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE
    phil2.running = FALSE
```

There are multiple available successors, let us take this successor which moves to the philosopher 3 going to the request left state, from thinking philosopher 3 moves to request left, this is state 0. From state 0 these are the possible successors let us choose 1 successor.

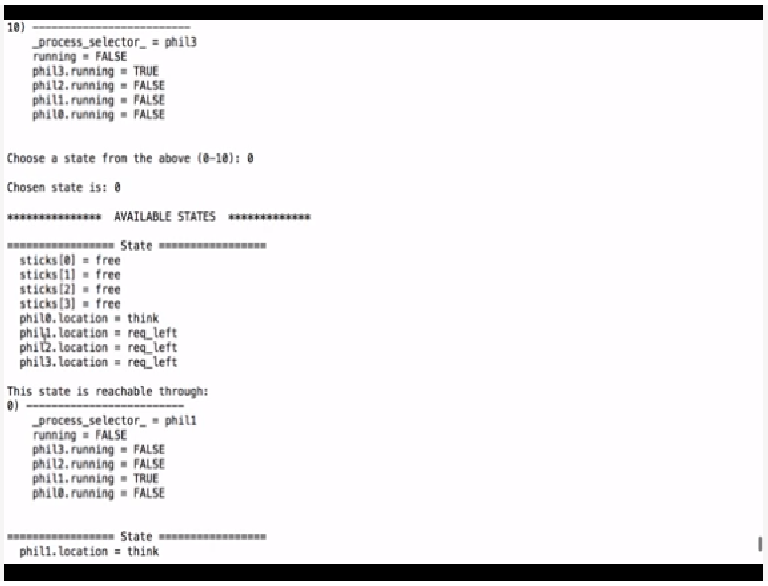
(Refer Slide Time: 19:35)



```
    phil1.running = FALSE
    phil0.running = TRUE


Choose a state from the above (0-12): 0

Chosen state is: 0

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = think
  phil2.location = req_left
  phil3.location = req_left

This state is reachable through:
0) -------------------------
    _process_selector_ = phil2
    running = FALSE
    phil3.running = FALSE
    phil2.running = TRUE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  phil2.location = think

This state is reachable through:
1) -------------------------
    _process_selector_ = phil0
    running = FALSE
```

Let us choose this successor where philosopher 2 is also moving to the request left state, this is successor 0.

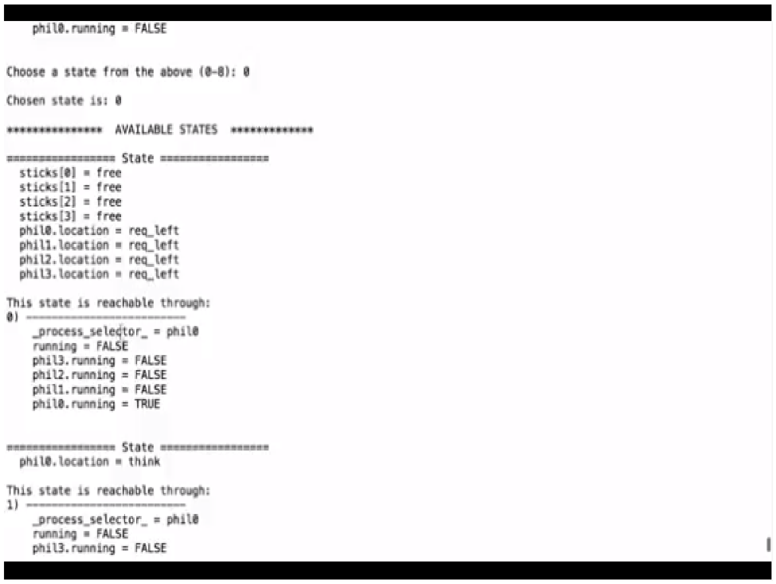
(Refer Slide Time: 19:50)



```
10) -------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


Choose a state from the above (0-10): 0

Chosen state is: 0

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = think
  phil1.location = req_left
  phil2.location = req_left
  phil3.location = req_left

This state is reachable through:
0) -------------------------
    _process_selector_ = phil1
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = TRUE
    phil0.running = FALSE


================= State =================
  phil1.location = think
```

Now, let us choose the successor where philosopher 1 moves to request left.

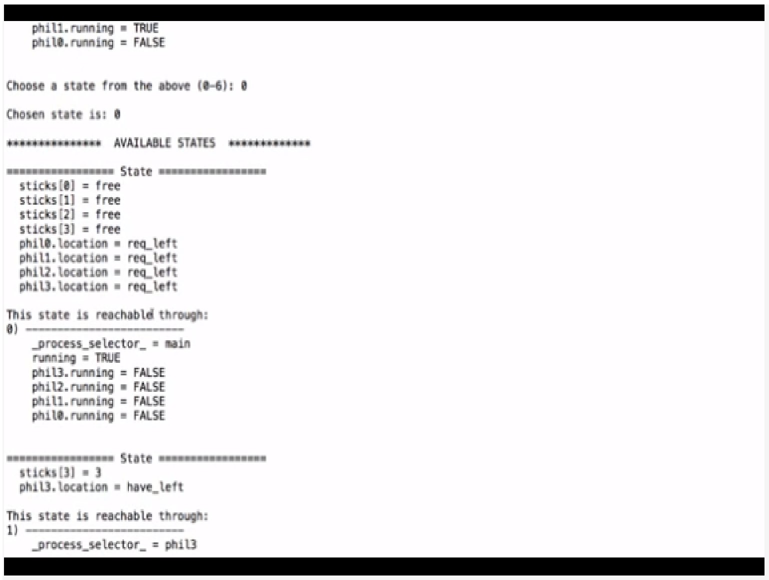
(Refer Slide Time: 20:01)



```
    phil0.running = FALSE


Choose a state from the above (0-8): 0

Chosen state is: 0

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = req_left
  phil1.location = req_left
  phil2.location = req_left
  phil3.location = req_left

This state is reachable through:
0) -------------------------
    _process_selector_ = phil0
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = TRUE


================= State =================
  phil0.location = think

This state is reachable through:
1) -------------------------
    _process_selector_ = phil0
    running = FALSE
    phil3.running = FALSE
```

And finally let us now choose the successor where philosopher 0 moves to request left. Right now all sticks are free and all the philosophers have moved to the request left state. Now what could the successors be.

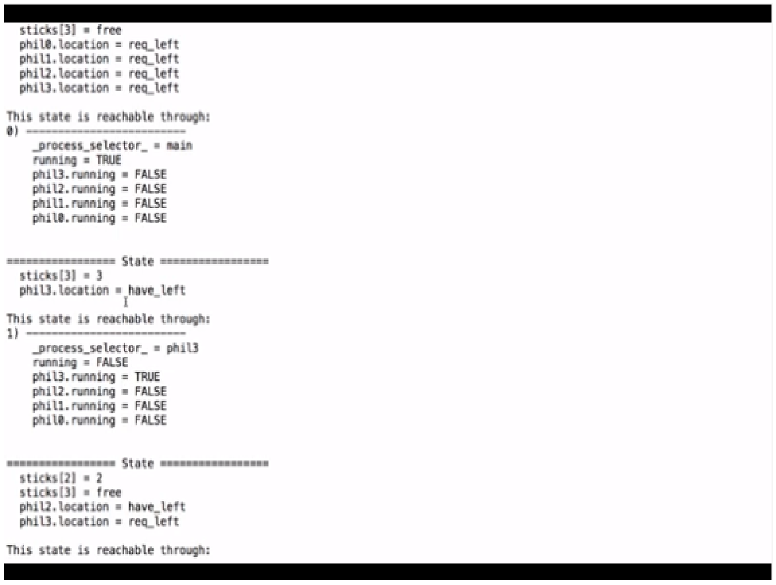
(Refer Slide Time: 20:26)



```
    phil1.running = TRUE
    phil0.running = FALSE


Choose a state from the above (0-6): 0

Chosen state is: 0

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = free
  phil0.location = req_left
  phil1.location = req_left
  phil2.location = req_left
  phil3.location = req_left

This state is reachable through:
0) -------------------------
    _process_selector_ = main
    running = TRUE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[3] = 3
  phil3.location = have_left

This state is reachable through:
1) -------------------------
    _process_selector_ = phil3
```

This says that if process main is selective then there is no change.

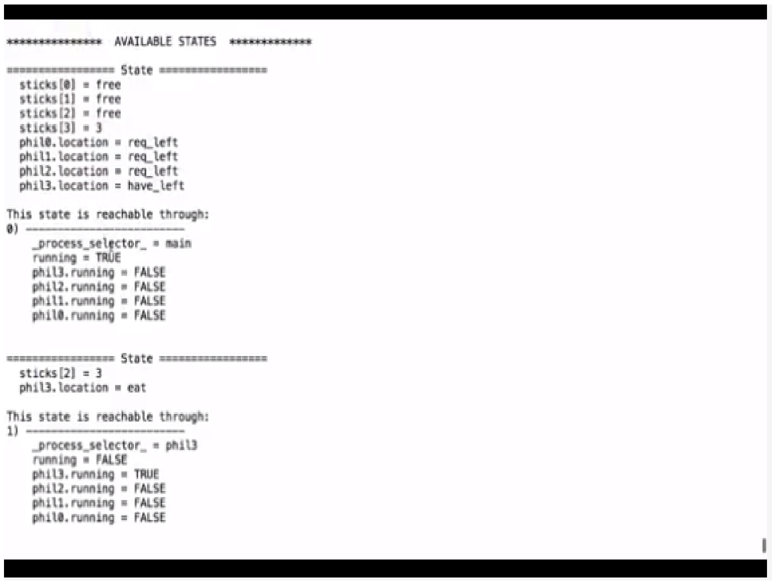
(Refer Slide Time: 20:34)



```
  sticks[3] = free
  phil0.location = req_left
  phil1.location = req_left
  phil2.location = req_left
  phil3.location = req_left

This state is reachable through:
0) -------------------------
    _process_selector_ = main
    running = TRUE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[3] = 3
  phil3.location = have_left

This state is reachable through:
1) -------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[2] = 2
  sticks[3] = free
  phil2.location = have_left
  phil3.location = req_left

This state is reachable through:
```

This is the state where philosopher 3 now moves to the have left because sticks of 3 is free philosopher 3 can have his left stick and then he sets sticks of 3 to be 3. Let us choose state 1,from state 1 there are multiple successors, in this successor okay let me explain this is the 1 where nothing has changed.

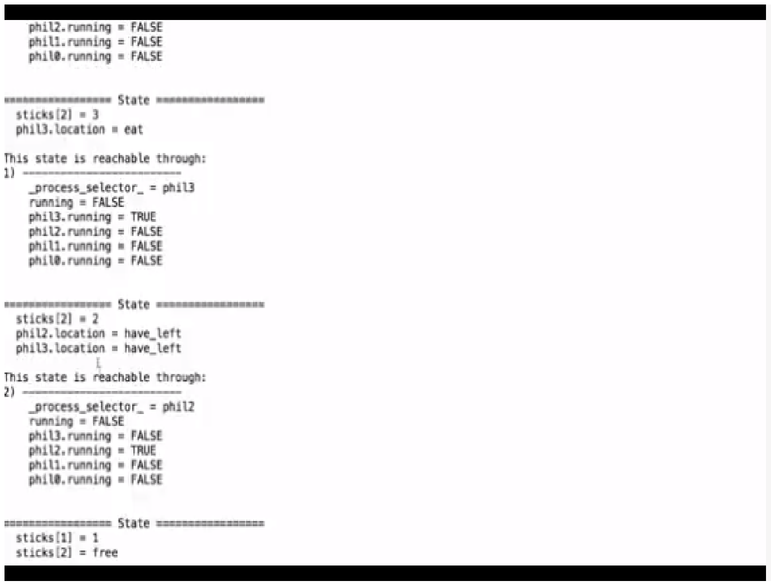
(Refer Slide Time: 21:09)



```
***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = free
  sticks[1] = free
  sticks[2] = free
  sticks[3] = 3
  phil0.location = req_left
  phil1.location = req_left
  phil2.location = req_left
  phil3.location = have_left

This state is reachable through:
0) -------------------------
    _process_selector_ = main
    running = TRUE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[2] = 3
  phil3.location = eat

This state is reachable through:
1) -------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE
```

This is the result of process main being selected, so none of the philosophers is executing his step.

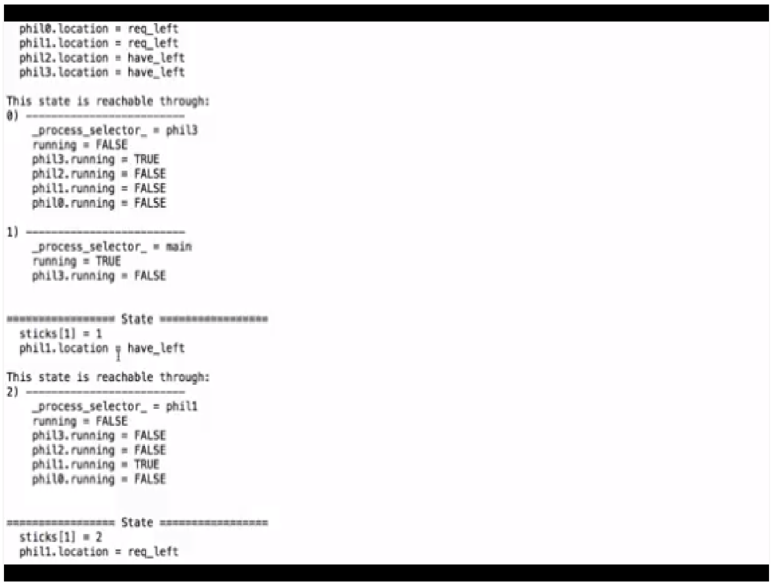
(Refer Slide Time: 21:21)



```
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[2] = 3
  phil3.location = eat

This state is reachable through:
1) ------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[2] = 2
  phil2.location = have_left
  phil3.location = have_left

This state is reachable through:
2) ------------------------
    _process_selector_ = phil2
    running = FALSE
    phil3.running = FALSE
    phil2.running = TRUE
    phil1.running = FALSE
    phil0.running = FALSE


================= State =================
  sticks[1] = 1
  sticks[2] = free
```

State 1 is when philosopher 3 goes to the eat state, state 2 is when philosopher 2 moves to have left. So, right now philosopher 3 is already in have left, philosopher 2 also moves to have left and he sets sticks of 2 to 2, this is state 2.

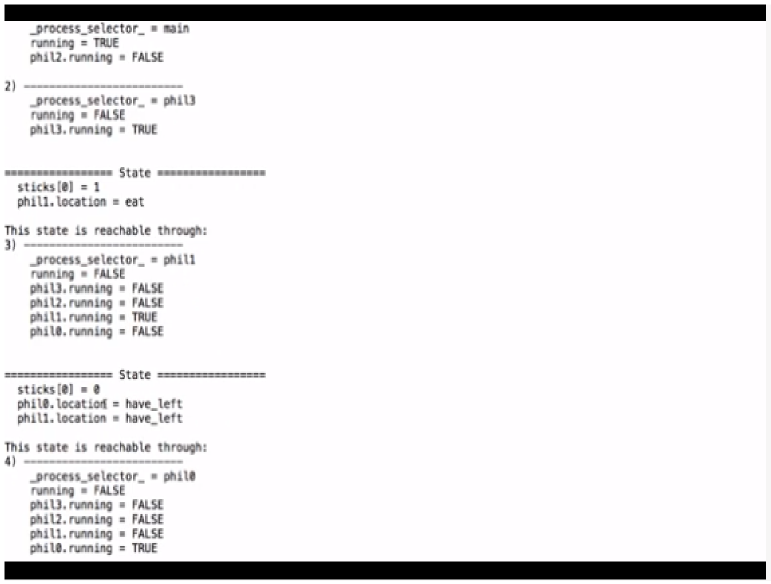
(Refer Slide Time: 21:48)



```
  phil0.location = req_left
  phil1.location = req_left
  phil2.location = have_left
  phil3.location = have_left

This state is reachable through:
0) ------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = FALSE

1) ------------------------
    _process_selector_ = main
    running = TRUE
    phil3.running = FALSE


================= State =================
  sticks[1] = 1
  phil1.location = have_left

This state is reachable through:
2) ------------------------
    _process_selector_ = phil1
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = TRUE
    phil0.running = FALSE


================= State =================
  sticks[1] = 2
  phil1.location = req_left
```

Now, from here let us choose the state where philosopher 1 moves to have left. Philosopher 3 is already in have left, philosopher 2 is in have left and philosopher 1 moves to have left as well, this was state 2.

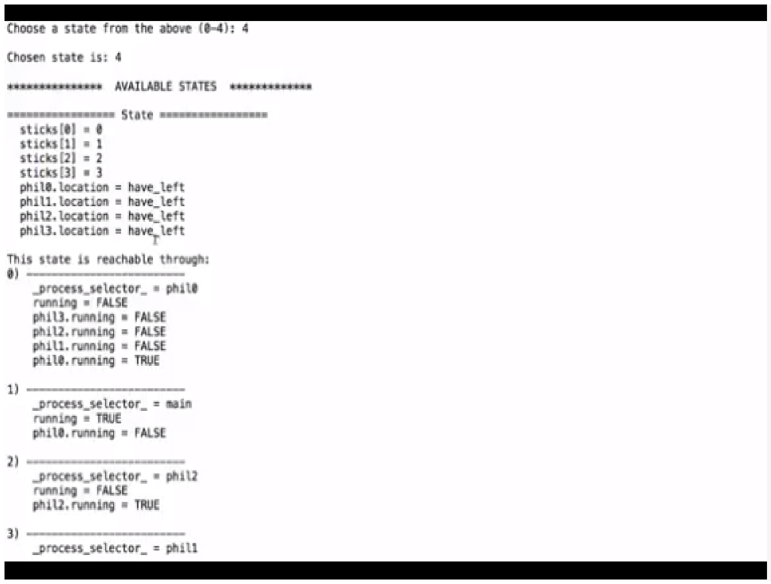
(Refer Slide Time: 22:14)



```
    _process_selector_ = main
    running = TRUE
    phil2.running = FALSE

2) --------------------------
    _process_selector_ = phil3
    running = FALSE
    phil3.running = TRUE


================= State =================
  sticks[0] = 1
  phil1.location = eat

This state is reachable through:
3) --------------------------
    _process_selector_ = phil1
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = TRUE
    phil0.running = FALSE


================= State =================
  sticks[0] = 0
  phil0.location = have_left
  phil1.location = have_left

This state is reachable through:
4) --------------------------
    _process_selector_ = phil0
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = TRUE
```

And now I will once again choose the situation where philosopher 0 takes his left stick sets it to 0, sticks of 0 to 0 and then moves to have left, this is state 4.

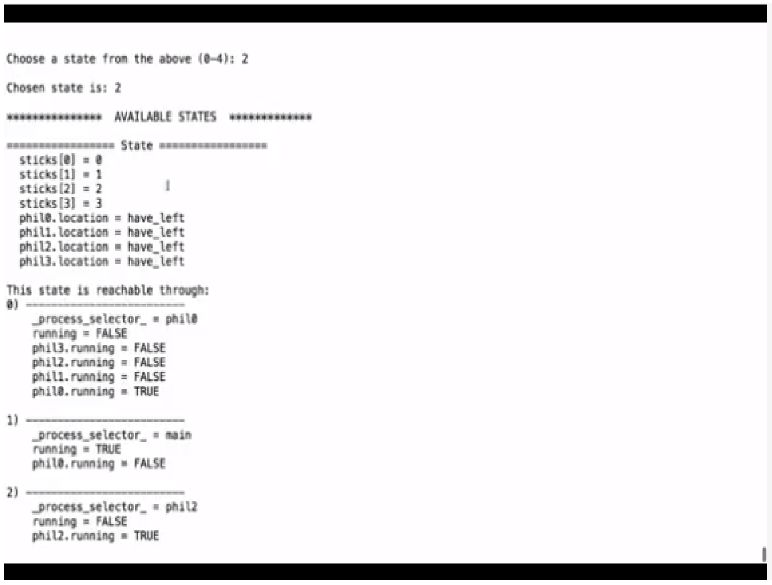
(Refer Slide Time: 22:23)



```
Choose a state from the above (0-4): 4

Chosen state is: 4

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = 0
  sticks[1] = 1
  sticks[2] = 2
  sticks[3] = 3
  phil0.location = have_left
  phil1.location = have_left
  phil2.location = have_left
  phil3.location = have_left

This state is reachable through:
0) --------------------------
    _process_selector_ = phil0
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = TRUE

1) --------------------------
    _process_selector_ = main
    running = TRUE
    phil0.running = FALSE

2) --------------------------
    _process_selector_ = phil2
    running = FALSE
    phil2.running = TRUE

3) --------------------------
    _process_selector_ = phil1
```

So, currently all the philosophers have taken their left stick. What could the possible successors b. There is only 1 successor which is the same state it just says that this is reachable through different selection of processes but no matter whatever process you select you get back to the same state. Let me choose say the process selector is philosopher 2 nothing happens.

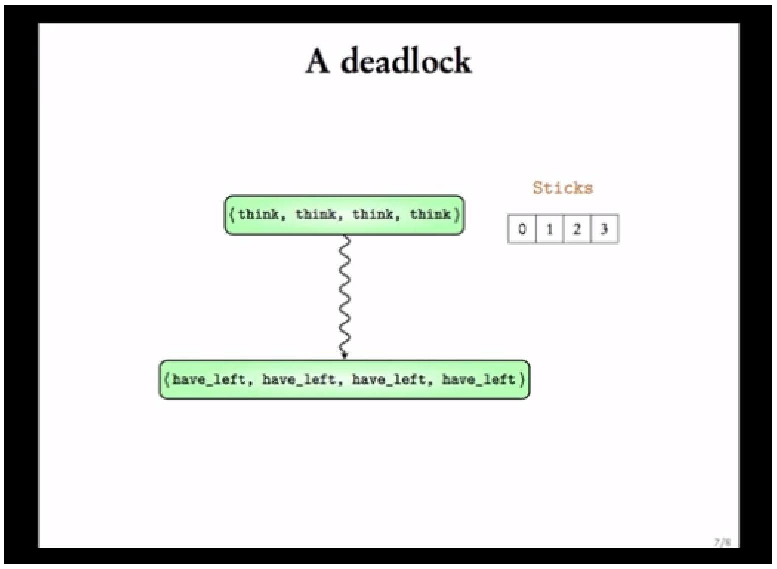
(Refer Slide Time: 23:09)



```
Choose a state from the above (0-4): 2

Chosen state is: 2

***************  AVAILABLE STATES  *************

================= State =================
  sticks[0] = 0
  sticks[1] = 1
  sticks[2] = 2
  sticks[3] = 3
  phil0.location = have_left
  phil1.location = have_left
  phil2.location = have_left
  phil3.location = have_left

This state is reachable through:
0) --------------------------
    _process_selector_ = phil0
    running = FALSE
    phil3.running = FALSE
    phil2.running = FALSE
    phil1.running = FALSE
    phil0.running = TRUE

1) --------------------------
    _process_selector_ = main
    running = TRUE
    phil0.running = FALSE

2) --------------------------
    _process_selector_ = phil2
    running = FALSE
    phil2.running = TRUE
```

If you see all of them have moved to a scenario where they have the left stick alone. Since, none of them can go to the eat state they cannot release the stick. So this is the situation where all of them keep waiting for the right stick, no matter whichever philosopher is chosen to proceed the state remains the same.

(Refer Slide Time: 23:49)



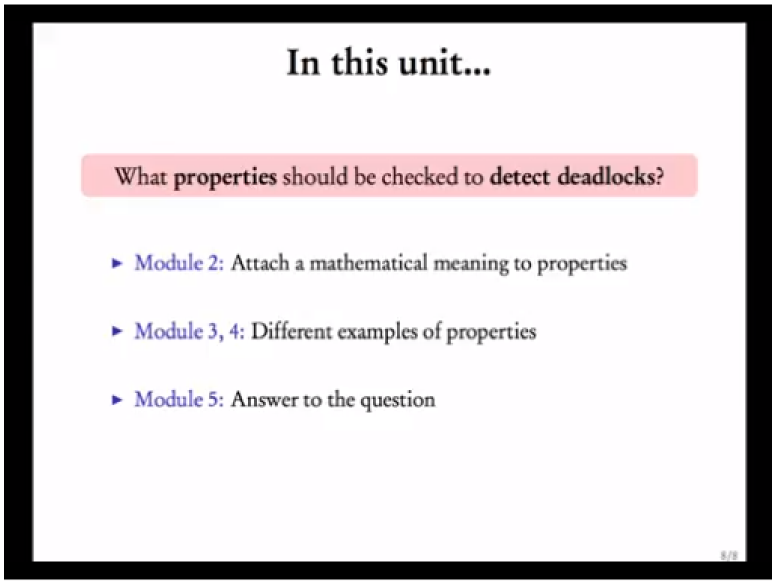
A deadlock

Sticks
0 1 2 3
(think, think, think, think)
(have_left, have_left, have_left, have_left)
7/8

If you see it always stays in the left, have left scenario this kind of a scenario is called a deadlock. From the initial state where all the philosophers were in the think state the system has reached a state where each of the philosopher processes is in the have left state and the sticks is like this; sticks of 0 contain 0, sticks of 1 contains 1, sticks of 2 contains 2, and sticks of 3 contains 3.

This process cannot move ahead because sticks of 1 is not free, similarly none of the processes can move ahead because the sticks are not free. Since none of the processes can move forward the situation is called a deadlock.

(Refer Slide Time: 26:26)



In this unit...

What **properties** should be checked to **detect deadlocks?**

- Module 2: Attach a mathematical meaning to properties
- Module 3, 4: Different examples of properties
- Module 5: Answer to the question

The question is, How do we detect such deadlocks in models? what we did now was a simulation I forged certain transitions and I showed that there is a deadlock scenario. But suppose you have a model in your hand and you want to check if such kind of deadlocks are present in the model what should you do?

This is going to be the subject of this unit, what properties should be checked to detect deadlocks. We will try to answer this question in a step by step manner. I hope that the question is clear, we saw this model of the dining philosophers it's a concurrent system the model had a deadlock. Now the question is, How do we detect such deadlock using NuSMV?

Are there certain properties that we can check instead of doing this manual simulation or there some properties that we can check so that the deadlocks are revealed. Now in the step by step procedure in module 2 we will first attach a mathematical meaning to the word properties.

In module 3 and 4 we will see different examples of properties they will still not be able to detect deadlocks. Finally in module 5 we will explain certain properties and ways of checking these properties in NuSMV which will reveal deadlocks.