11212832罗傲雪Lab 1: PROMELA/SPIN

**Part 1 – The dining philosophers**

**Q1.** Use assertions in your code to verify this property about forks.

**Q2.**

**a.** Does your model have deadlocks? If yes, use SPIN to derive a counter example showing a

problematic trace. Modify your solution to eliminate deadlocks.

**b.** Verify absence of deadlocks in your model using SPIN.

**Q3.** Verify property (1) with an LTL formula instead of assertions

**A:**

**A1:**

Related files are:

Dining\Dining.pml : the code of my first try for Dining problem;

Dinig\Dinig\_result.txt : the result after verification for Dining\Dining.pml;

Dining\Ding\_trail\_output.txt : the counter example for the verification.

assertions in my code are listed below:

|  |
| --- |
| Code-1.1 |
| /\*assertions to verify safety to ensure that a fork can not be used by two philosopher simultaneously \*/  assert(forks[id]==1);  assert(forks[(id+1)%NUM\_PHIL]==1); |

And the main idea of my first try of Dining\Dinign.pml is :For each philosopher he will firstly get its left fork then for its right fork.

**A2-a:**

After verification for “*spin –a Dining.pml && gcc –DSAFETY pan.c –o pan && .\pan >Dinig\_result.txt”.* the result is showed in Dining\_result.txt .we found that there exist invalid end state .So the model of my first try for the Dining problem have deadlock problem. simulate Dining.pml in guided option by “spin –t –p –l Dining.pml >Dining\_trail\_output.txt”. So the counter example is showed in Dining\_trail\_output.txt.

|  |
| --- |
| Cut from Dining\_trail\_output.txt |
| #processes: 5  forks[0] = 1  forks[1] = 1  forks[2] = 1  forks[3] = 1  76: proc 4 (phil) Dining.pml:20 (state 11)  76: proc 3 (phil) Dining.pml:20 (state 11)  76: proc 2 (phil) Dining.pml:20 (state 11)  76: proc 1 (phil) Dining.pml:20 (state 11)  76: proc 0 (:init:) Dining.pml:44 (state 9) <valid end state> |

As showed in the table. At last, all philosophers are blocked in line-20 ,which is to fetch the right fork. So at last , all philosopher can not eat as they all hold the left fork without release the right.

A2-b

After modification, the related code are:

Dining\Dining1.pml : the code of my second try for the Dining problem;

Dining\Dining1\_result.txt: the verification result of Dining\Ding1.pml;

From Dining1\_result.txt. we found that there exist no error. so deadlock is avoided in my second try for Dining problem.

|  |
| --- |
| Cut from Dining\Dining1\_result.txt |
| State-vector 88 byte, depth reached 483, errors: 0 |

The main idea of Dining\Dining1.pml is : firstly, we number the forks with distinguished identifier. And for each philosopher, one can only fetch the fork with lower identifier then for the higher identifier. And the partial order of the forks help to ensure the avoid of deadlock.

3-A:

Simply delete the assertion statement. LTL is as below:

ltl safety{[]((forks[0]<=1)&&forks[1]<=1 &&forks[2]<=1)}

**Part 2 –Needham-Schroeder protocol verification**

**Task 1 && Task 2:**

Related files:

NS2.pml: adding code for Bob

NS2\_result.txt: the verification result of NS2.pml

Add related code for Bob follow the protocol descript in the problem. and verified by

Spin –a NS2.pml && gcc –DNOREDUCE –o pan.c pan && .\pan –a >NS2\_result.txt

ltl BOTH\_ARE\_OK: <>(((statuA==ok)) && ((statuB==ok)))

After verification ,found that, error:0. Means both status of Alice and Bob can reach ok.

**Task 3 :**

Related files:

NS3.pml: the code that introduce the intruder;

NS3\_result.txt: the verification result of NS3.pml;

NS3\_trail\_output.txt: the output of simulate NS3 in guided option.

After introduce the intruder, found there is error:1

and the counter example shows in NS3\_trail\_output.txt

|  |
| --- |
| Cut from NS3\_trail\_output.txt |
| 25: proc 2 (Intruder) NS3.pml:143 (state 33)  25: proc 1 (Bob) NS3.pml:80 (state 9)  25: proc 0 (Alice) NS3.pml:36 (state 6) |
|  |

And when execute in ispin. Get the communication graph which can help to analysis the counter example.

The execution is pended in

Line-143 for intruder (network ! msg(recpt,data);)

Line-80 for Bob ((data.key==keyB)&&(data.content1==partnerB);)

Line-36 for Alice (network ! msg1(partnerA, messageAB);)

So the execution is , if intruder send message first. Then after Bob received it, will be blocked in checking the key or content1. And Alice will be blocked in sending message for reason that no receiver in the network. As showed in NS3\_trail\_output.txt

partnerA = agentB

partnerB = agentA

statuA = err

statuB = err

the property can not be satisfied.

**Task 4 && Task 5 &&Task 6**:

Related files:

NS6.pml: modified some code for intruder and Alice so is Bob.

NS6\_result.txt: the verification result of NS6.pml for the property of ltl PROP\_AB

NS6\_result1.txt: the verification result of NS6.pml for the property of ltl PROP\_A

NS6\_result2.txt: the verification result of NS6.pml for the property of ltl PROP\_B

The Ltl property and the modified code listed below:

|  |
| --- |
| ltl PROP\_AB {[]((statusA == ok && statusB == ok) -> (partnerA == agentB && partnerB == agentA))}  ltl PROP\_A {[]((statusA == ok && partnerA == agentB) -> (!knows\_nonceA))}  ltl PROP\_B {[]((statusB == ok && partnerB == agentA) -> (!knows\_nonceB))} |
| Alice:  if  ::true->partnerA = agentB;  pkey = keyB;  ::true->partnerA = agentI;  pkey = keyI;  fi; |
| Bob:  if  ::true->partnerB = agentA;  pkey = keyA;  ::true->partnerB = agentI;  pkey = keyI;  fi; |

After verification ,we found that only PROP\_A hold in NS6.pml.

|  |
| --- |
| Cut from NS6\_trail\_output.txt |
| #processes: 3  partnerA = agentI  partnerB = agentA  statusA = ok  statusB = ok  knows\_nonceA = 1  knows\_nonceB = 1  84: proc 2 (Intruder) NS6.pml:177 (state 49)  Intruder(3):intercepted.key = 0  Intruder(3):intercepted.content1 = 0  Intruder(3):intercepted.content2 = 0  Intruder(3):data.key = keyB  Intruder(3):data.content1 = nonceB  Intruder(3):data.content2 = 0  84: proc 1 (Bob) NS6.pml:111 (state 19) <valid end state>  Bob(2):data.key = keyB  Bob(2):data.content1 = nonceB  Bob(2):data.content2 = 0  Bob(2):messageBA.key = keyA  Bob(2):messageBA.content1 = nonceA  Bob(2):messageBA.content2 = nonceB  84: proc 0 (Alice) NS6.pml:60 (state 21) <valid end state>  Alice(1):data.key = keyA  Alice(1):data.content1 = nonceA  Alice(1):data.content2 = nonceB  Alice(1):messageAB.key = keyI  Alice(1):messageAB.content1 = nonceB  Alice(1):messageAB.content2 = 0 |
|  |

It’s a counter example to verify PROP\_B. from the communication graph is easy to understand what happened.

**Task 7:**

Related files:

NS7.pml: final version of NS.

NS7\_result.txt: the verification result of NS7.pml for the property of ltl PROP\_AB

NS7\_result1.txt: the verification result of NS7.pml for the property of ltl PROP\_A

NS7\_result2.txt: the verification result of NS7.pml for the property of ltl PROP\_B

From the result files for NS7 ,conclude that all verifications are satisfied.