

# Modeling and Formal Verification of The Dining Philosophers Problem Using SPIN

by

**Kripalsinh Makvana**  
**202011054**

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

MASTER OF TECHNOLOGY

in

INFORMATION AND COMMUNICATION TECHNOLOGY

to

**DHIRUBHAI AMBANI INSTITUTE OF INFORMATION AND COMMUNICATION TECHNOLOGY**



June, 2022

## Declaration

I hereby declare that

- i) the thesis comprises of my original work towards the degree of Master of Technology in Information and Communication Technology at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,
- ii) due acknowledgment has been made in the text to all the reference material used.



---

Kripalsinh Makvana

## Certificate

This is to certify that the thesis work entitled **MODELING AND FORMAL VERIFICATION OF THE DINING PHILOSOPHERS PROBLEM USING SPIN** has been carried out by **KRIPALSINH MAKVANA (202011054)** for the degree of Master of Technology in Information and Communication Technology at *Dhirubhai Ambani Institute of Information and Communication Technology* under my supervision.



---

Prof. Puneet Bhateja  
Thesis Supervisor

# Acknowledgments

I want to express my gratitude to Dr.Puneet Bhateja my thesis supervisor for his continuous support and advice throughout the program. Working with him is a great honour for me. His pleasant personality, willingness to help, and superior subject knowledge have all helped me to bring out the best in me. Without his guidance, doing this work would have been extremely difficult.

Finally, I want to thank my parents for believing in me and supporting me. Thank you for your blessings and motivation.

# Contents

<b>Abstract</b>	<b>iv</b>
<b>List of Figures</b>	<b>v</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Thesis Objectives and Problem Description . . . . .	2
1.2 Contributions . . . . .	2
1.3 Thesis Organization . . . . .	3
<b>2 The basic structure of the SPIN model checker</b>	<b>4</b>
2.1 The basic structure of the SPIN . . . . .	5
2.2 The essential components of the PROMELA . . . . .	5
2.3 Verification in SPIN . . . . .	6
2.4 Summary . . . . .	6
<b>3 The first solution of the dining philosophers problem</b>	<b>7</b>
3.1 Approach 1 . . . . .	8
3.2 PROMELA model . . . . .	8
3.3 Problems . . . . .	9
3.4 Summary . . . . .	10
<b>4 The second solution of the dining philosophers problem</b>	<b>11</b>
4.1 Approach 2 . . . . .	12
4.2 PROMELA model . . . . .	12
4.3 Analyzing and Verification of the model . . . . .	14
4.4 Summary . . . . .	15
<b>5 Conclusion</b>	<b>16</b>
<b>References</b>	<b>17</b>

# Abstract

The SPIN tool is used for verifying the correctness of the system. SPIN stands for simple PROMELA interpreter. It's been used to find design problems in systems. The main idea behind the thesis is to provide an overview of the spin model's architecture and functionality using the dining philosophers problem. The dining philosophers problem is a standard synchronization-related concurrency problem. One solution to the problem is chosen for modeling and verification. For modeling, PROMELA is used, and SPIN is used for verification.

SPIN can verify the model by running random simulations or creating a C program. It can do a complete verification to determine whether the system's behavior is error-free or not with mathematical certainty. PROMELA is used for modeling system models in formal verification that allows for the dynamic development of concurrent processes and identifies interactions between processes in a distributed system. We examined and validated various properties of the Dining Philosophers Problem, such as the absence of deadlock and the absence of individual starvation. Using simulation, we determined the expected execution of the solution. There were no invalid end-states or non-progress cycles discovered.

# List of Figures

1.1	Philosophers sitting around food. [1] . . . . .	1
2.1	The basic structure of the SPIN. G. J. Holzmann. The model checker spin. IEEE Transactions on software engineering, 23(5): 279–295, 1997 Page No- 280. [2] . . . . .	4
3.1	States of the philosopher and forks for solution 1. C. Baier and J.-P. Katoen. Principles of model checking. MIT Press, 2008. Page No- 92. [1] . . . . .	7
4.1	States of the philosopher and forks for solution 2. C. Baier and J.-P. Katoen. Principles of model checking. MIT Press, 2008. Page No- 93. [2] . . . . .	11

# CHAPTER 1

## Introduction

Formal verification is the act of checking model specifications for flaws that could result in incorrect operation. Formal verification is better than testing because it can prove the absence of errors such as deadlock, livelock, etc.; it is entirely automated, less expensive to fix the errors, and is independent of implementation.[4] The goal of SPIN verification models is to prove the correctness of process interactions by abstracting as much as possible from internal sequential calculations. SPIN generates C sources code for a model checker rather than performing model checking itself. This method saves memory and increases efficiency while also allowing sections of C code to be directly inserted into the model. The notations in SPIN are selected such that the tool can mechanically demonstrate the logical consistency of a design.

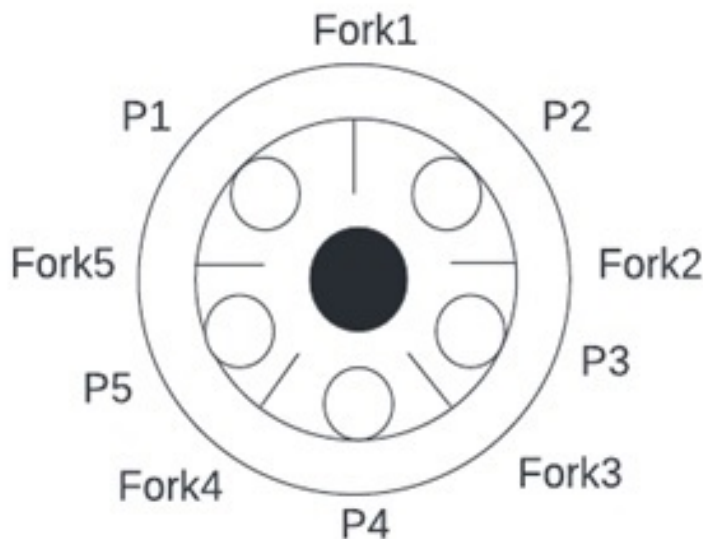


Figure 1.1: Philosophers sitting around food. [1]

# 1.1 Thesis Objectives and Problem Description

Dijkstra created the Dining Philosophers Problem in 1965 to demonstrate the horror that is deadlock. In Figure 1.1, one can observe that, A group of five philosophers sits around a table. A philosopher's sole goal in life is to think, yet in order to feed their thinking body, they occasionally have to take some time off to eat. For this reason, the table has in its center a large bowl of spaghetti. Unfortunately, because of their university department's tight budget, there are only as many forks as there are philosophers. However, because they are not of Italian descent, none of them has mastered the technique of eating spaghetti with only one fork; each philosopher eats with exactly two forks. They agree to place each fork between two philosophers so that each fork is shared between two of them.

When a philosopher gets hungry, he seeks to acquire the forks to his left and right one at a time. It is non-deterministic which side he begins on. After receiving two forks, they will eat for a while, then return their forks and return to thinking.

In this thesis, we use the SPIN(Simple PROMELA Interpreter) model checker because SPIN is freely available on the Internet and is the world's most widely used logic-based model checker. Bell Labs developed it and has been awarded the "Software System Award" by the ACM (Association for Computing Machinery). SPIN can validate most of the models that are written in PROMELA(PROcess MEta LAnguage), so we use PROMELA as a modeling language and SPIN as a simulator and verifier in our thesis. [2] The dining philosophers problem is a classic example problem in concurrent algorithm design for displaying synchronization problems and possible solutions.

The Dining Philosophers problem demonstrates the difficulties of handling shared states in a multithreaded environment. In this thesis, we model the dining philosophers problem in PROMELA, verify that model in SPIN and check if there are any invalid end-states(deadlock) or non-progress cycles discovered or not.

## 1.2 Contributions

The following is the thesis contribution :

- **Modeling and formal verification of dining philosophers problem.** We



Modeled a simple solution of the dining philosophers problem and verified that model in SPIN. After verification, we found two errors in that approach, so we changed that approach a little bit and modeled the second solution, and verified it in SPIN without getting any errors.

## 1.3 Thesis Organization

The thesis report is further divided into the following chapters:

- Chapter 2 presents the basic structure of the SPIN model checker.
- Chapter 3 presents the first approach for solving the dining philosophers problem.
- Chapter 4 presents the second approach for solving the dining philosophers problem.

## CHAPTER 2

# The basic structure of the SPIN model checker

In this chapter, we show the basic structure of the SPIN model checker and see how it works and see some basics of PROMELA modeling Language.

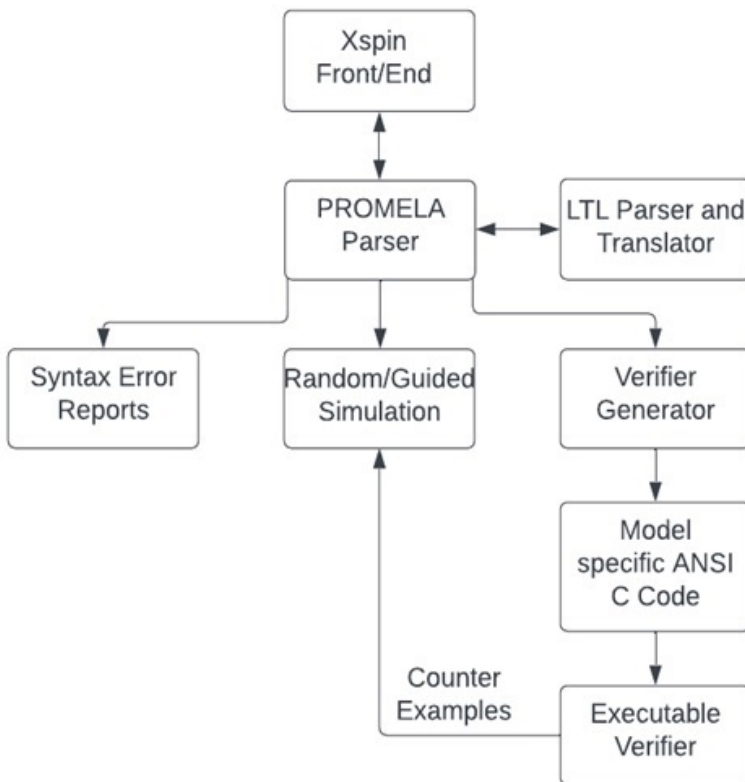


Figure 2.1: The basic structure of the SPIN. G. J. Holzmann. The model checker spin. IEEE Transactions on software engineering, 23(5): 279–295, 1997 Page No-280. [2]

## 2.1 The basic structure of the SPIN

Figure 2.1 represents the basic structure of the SPIN model checker. Here xspin is a graphical front-end of the SPIN model checker. ispin is written in Tcl/tk. Tcl is a high-level, general-purpose, dynamic programming language.[3]

SPIN takes PROMELA code compile and checks for syntax errors and gives a report in the first step. In the second step, interactive simulation is run until fundamental confidence in the design's behavior is obtained.

And in the third step, SPIN Verifier Generator compiles the code into c, and then If a counterexample to the claims of correctness is discovered. These may be input back into the interactive simulator and examined in depth to determine and eliminate the source of the problem.

## 2.2 The essential components of the PROMELA

Processes, message channels, and variables are the three categories of objects in SPIN models. Processes are global objects. Within a process, message channels and variables can be defined either globally or locally. Processes determine behavior, whereas channels and global variables create the context in which they operate.

The PROMELA model consists to a declaration of type, a declaration of the channel, a declaration of a variable, and a declaration of the process. A process type consists of a name, a list of formal parameters, declarations of local variables, and the process body. The body is made up of a series of sentences. Processes may be created with the run command, which returns the process id; alternatively, active can be included before the proctype declaration to generate processes.

A variable declaration begins with a keyword specifying the variable's data type, such as bit, bool, byte, short, or int, followed by one or more identifiers, and optionally an initializer: bit, bool, byte, short, or int. byte a, byte b = 4 If one is supplied, the initializer must be a constant. Variables are initially set to zero by default.

## 2.3 Verification in SPIN

In verification spin, check for if there any INVALID END-STATE, any NON-PROGRESS CYCLE, or any NEVER CLAIMS are present.

Valid end-states are those in which the process instance and the init process have either completed or are stuck at a statement in their defining program body with a label that begins with the prefix 'end.' End-state that is valid must also have empty channels. The remaining states are all invalid end-states.

A non-progress cycle (NPC) is an execution that does not loop through a progress state infinitely. With PROMELA, a simple verification of 'tasks' or requirements can be modeled as never claims, and Spin can swiftly prove or disprove that claim using the Linear Time Temporal logic (LTL) formula.[2]

## 2.4 Summary

In this chapter, we explain the basic structure of the SPIN model checker, how it works, and some essential components of the PROMELA modeling language. We also explain how SPIN verifies the model that is written in PROMELA.

## CHAPTER 3

# The first solution of the dining philosophers problem

In this chapter, we solve the dining philosophers problem with a simple approach, create a model for that solution, and verify the model for checking the presence of any errors.

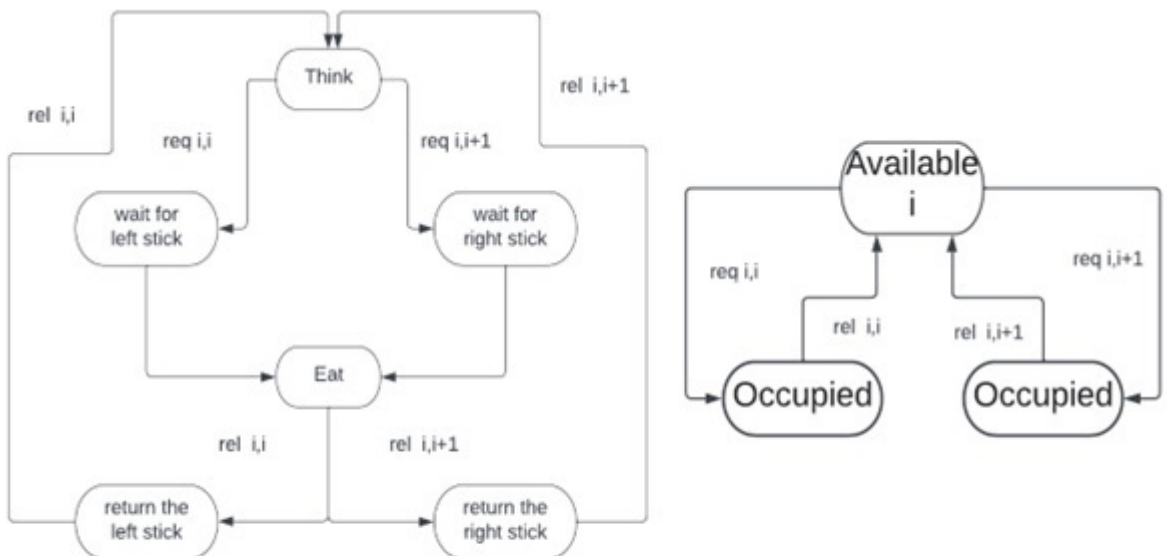


Figure 3.1: States of the philosopher and forks for solution 1. C. Baier and J.-P. Katoen. Principles of model checking. MIT Press, 2008. Page No- 92. [1]

## 3.1 Approach 1

The first approach for a problem to get and release forks is as follows.

While the philosopher is in a thinking state and wants to eat, then first philosopher picks the left fork, if it is available, then the right fork. After eating, the philosopher can quickly release both forks, as shown in Figure 3.1. Figure 3.1 also shows the states of the fork. When the philosopher requests the fork, the fork goes into an occupied state, and when the philosopher releases the fork, the fork goes back to the available state. Note that in this method, any philosopher can request the fork.

## 3.2 PROMELA model

```
#define PHIL 5
mtype={fork}
#define le forks[no]
#define ri forks[(no+1)% PHIL]
chan forks[PHIL] = [1] of {int};

proctype p(int no)
{
    do
        ::le?fork->ri?fork;
            le!fork;
            ri!fork
    od
}

init
{
    int philoso=PHIL;
    atomic
    {
        do
            ::philoso>0->philoso--;
                run p(philoso);
                forks[philoso]!fork
        od
    }
}
```

```

    ::philoso==0->break
od
}
}

```

Here PHIL is defined as 5, fork is the mtype variable, lf and rf are channels. There are five channels of int with a capacity of one each. The execution starts from the init process. First process p is run with the argument philoso-1, and one is sent to channel forks[philoso-1]. If the philoso variable is zero, then the break statement is executed.

In proctype p first fork receive one from channel lf, which is similar to picking up the left fork if it is available; if fork receives one from the lf, then it does the same operation with channel rf, same as picking right fork and then after some time send one to both channel one by one same as putting both forks on the table after eating. This whole process runs infinitely.

### 3.3 Problems

After verifying the above model in SPIN, we found two problems with this approach.

The most prominent issue with this program is the possibility of a deadlock. What if all philosophers sat down at the exact moment and picked up their left fork simultaneously?

In this case, all forks are locked, and none of the philosophers can successfully lock his right fork. As a result, we get circular waiting; each philosopher waits for his right fork, which his right neighbor is now locking, and a deadlock arises.

The second problem is starvation. Suppose two philosophers are both fast thinkers and fast eaters. They think quickly and get hungry fast. They then take their seats in opposing chairs. They may be able to choose their forks and eat since they are so fast. After they finish eating and before their neighbors can pick the fork and eat, they return again and pick the forks and eat.

In this situation, even though the other three philosophers have been seated for a long time, they have no opportunity to eat. This is known as starvation. It is important to note that this is not a deadlock because there is no circular waiting, and everyone gets a chance to eat.

### **3.4 Summary**

In this chapter, we discuss a simple solution to the dining philosophers problem; we modeled it in PROMELA, verified it in SPIN, and found that this approach has two problems.



## CHAPTER 4

# The second solution of the dining philosophers problem

In this chapter, we solve the dining philosophers problem with an improved approach, create a model for that solution, and verify the model for checking the presence of any errors.

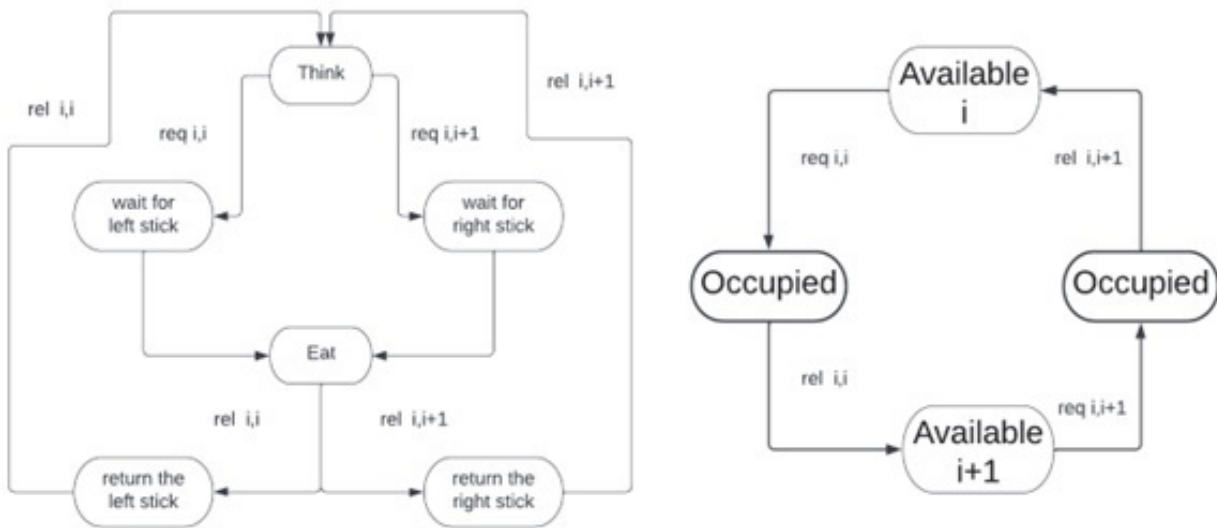


Figure 4.1: States of the philosopher and forks for solution 2. C. Baier and J.-P. Katoen. Principles of model checking. MIT Press, 2008. Page No- 93. [2]

## 4.1 Approach 2

The main issue with Approach 1 is that a single fork is desired by two philosophers simultaneously; however, if just one philosopher may access the fork at a time while the other philosopher waits for its turn, we can address this problem.

The forks in this approach have two states: red and blue. In the red state, fork  $i$  can only be used by philosopher  $i$ , and philosopher  $i+1$  cannot use it. In the blue state, fork  $i$  can only be used by philosopher  $i+1$ , and philosopher  $i$  cannot use it. After eating, the forks' state will shift from red to blue or blue to red as shown in Figure 4.1.

The initialization states of the forks are fixed in this approach. The initialize state of the fork  $i$  can be red or blue, but if the initialize state is red for fork  $i$ , then for fork  $i+1$ , it will be blue or vice versa; when the philosopher has done eating, the forks' states change.

If the states are initialized with blue, there are three blue and two red forks, and vice versa. Note that in this method, only two philosophers can eat simultaneously; others must wait for their turn, and when the philosopher has satisfied his hunger, he puts down both forks, so the states of the forks have to change, and other philosophers can eat. So after a few turns, all forks return to the initial state, and this cycle restarts again.

## 4.2 PROMELA model

```
#define PHIL 5
mtype={fork}
#define lf forks[a]
#define rf forks[(a+1)%PHIL]
chan forks[PHIL] = [1] of {int};
int i=1;

proctype p(int a)
{
    do
```

```

:: if
::i==1&&(a==2||a==4) ->
    lf?fork->rf?fork;
    lf!fork;
    rf!fork
    i++

::i==2&&(a==3||a==1) ->
    lf?fork->rf?fork;
    lf!fork;
    rf!fork
    i++

::i==3&&(a==2||a==5) ->
    lf?fork->rf?fork;
    lf!fork;
    rf!fork
    i++

::i==4&&(a==1||a==4) ->
    lf?fork->rf?fork;
    lf!fork;
    rf!fork
    i++

::i==5&&(a==5||a==3) ->
    lf?fork->rf?fork;
    lf!fork;
    rf!fork
    i++

::i==6||i>6->i=1;

::else->i++;
fi
od
}

```

```

init{
    int philoso=PHIL;
atomic {
    do
        ::philoso>0->philoso--;
        run p(philoso);
        forks[philoso]!fork
        ::philoso==0->break
    od
        }
    }
}

```

Here PHIL is defined as 5, fork is the mtype variable, lf and rf are channels. There are five channels of int with a capacity of one each. The execution starts from the init process. First process p is run with the argument philoso-1, and one is sent to channel forks[philoso-1]. If the philoso variable is zero, then the break statement is executed.

In proctype p it checks for i and a variables. Receive and send operation is only executed where if condition is fulfilled; for example, if i is equal to two and a is equal to one or three, then only receive and send operation will happen, and then i is increased by one, and if i is equal to or greater than six, then it's value change to one. This whole process runs infinitely.

### 4.3 Analyzing and Verification of the model

The main issue with Approach 1 is that a single fork is desired by two philosophers simultaneously. So if just one philosopher may access the fork at a time while the other philosopher waits for its turn, then this problem can be addressed.

In the second approach, only one philosopher can access the fork at a time, and the solution is based on turns no one can pick forks if it is not there turn so this way we solve both problems of deadlock and individual starvation.

INVALID END-STATE(DEADLOCK): In PROMELA, valid end-states are system states in which all process instances and the init process have either completed their defining program body or are stuck at a statement with the label beginning with the prefix 'end.' Valid end-states require empty channels. All of the other states are invalid end-states.[2]

SPIN did not find any errors like invalid end state, assertion violation, or acceptance cycle during verification of the second solution. So we can verify that the second solution is error-free.

## 4.4 Summary

In this chapter, we discuss an improved solution to the dining philosophers problem; we modeled it in PROMELA, verified it in SPIN, and explain how this solution solves the problem of deadlock and individual starvation.

## CHAPTER 5

# Conclusion

Most well-developed engineering fields contain a process for building and analyzing design prototypes. SPIN is a tool for distributed systems design to develop the basic methodology for on-the-fly automated verification.

We modeled the original dining philosophers problem and demonstrated that the system is vulnerable to deadlock. The design was then modified slightly, demonstrating that the system was free of deadlock and starvation using SPIN.

The work also demonstrates how simple it is in PROMELA to model any problem and the necessary corrective claims. SPIN has also been discovered to be quite useful and efficient in the formal verification of the PROMELA model.

# References

- [1] C. Baier, J. Katoen, and K. Larsen. *Principles of Model Checking*. MIT Press, 2008.
- [2] G. J. Holzmann. The model checker spin. In *IEEE Transactions on software engineering*, volume 23, pages 279–295, 1997.
- [3] M. J. Hornos and J. C. Augusto. Installation process and main functionalities of the spin model checker. In *Electronic version available at <http://digibug.ugr.es/handle/10481/19601>*. Universidad de Granada, 2012.
- [4] S. M. Islam, M. H. Sqalli, and S. Khan. Modeling and formal verification of dhcp using spin. In *Int. J. Comput. Sci. Appl.*, volume 3, pages 145–159, 2006.