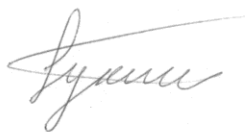


Минобрнауки России  
Санкт-Петербургский государственный политехнический университет  
Институт информационных технологий и управления  
Кафедра «Информационные и управляющие системы»

## КУРСОВАЯ РАБОТА

Разработка контроллера светофоров и его верификация  
по дисциплине «Распределенные алгоритмы и протоколы»

Выполнил  
студент гр. 6084/12



А.Лукашин

Руководитель  
ст. преподаватель

Шошмина И.В.

«10» декабря 2013 г.

Санкт-Петербург

2013

## Оглавление

Введение .....	3
Постановка задачи.....	3
Реализация.....	4
Основная идея.....	4
Модель на языке Promela.....	4
LTL правила .....	7
Безопасность .....	7
Живость и справедливость .....	8
Результаты моделирования .....	9
Выводы .....	10
Список литературы .....	10

## Введение

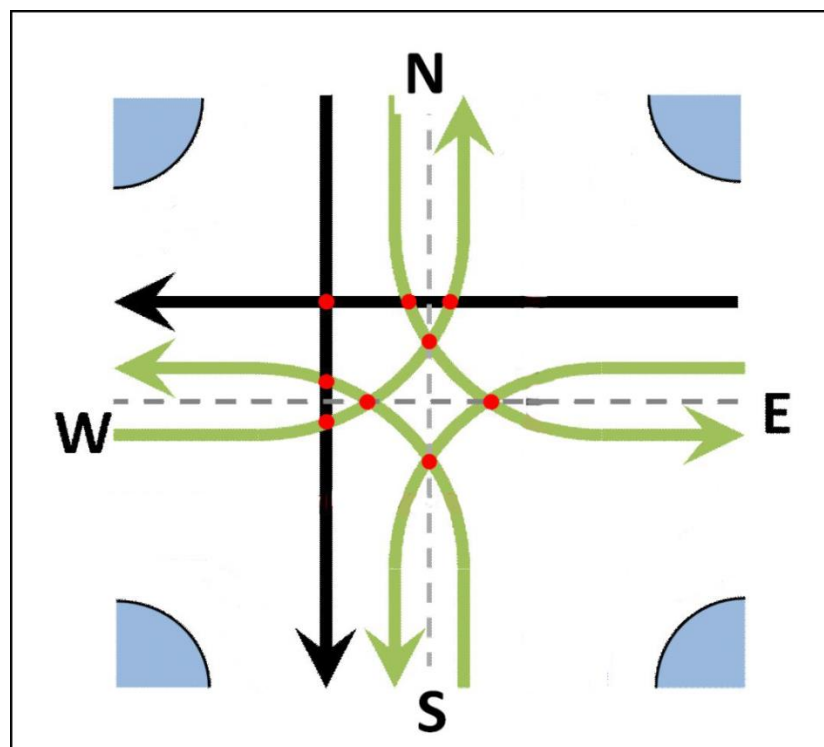
Верификация – это проверка модели (алгоритма, системы) на корректность работы, с учетом заданных правил. В настоящее время вопрос корректности работы программных продуктов является крайне актуальным. На программное обеспечение возлагаются все новые и новые функции, при это в некоторых областях возникновение ошибки может приводить к катастрофическим последствиям. Такие тенденции позволяют говорить о важности применения и развития верификационных средств.

В данной курсовой работе рассматривается модель управления движением на перекрестке, при условии, что каждое направление контролируется своим светофором. При этом для model checking используется система Spin. Данная система позволяет описывать модель на языке Promela и задавать требуемые свойства системы при помощи выражений LTL. Такой подход позволяет проверить корректность модели перед ее реализацией.

## Постановка задачи

Вариант (1, 12, 15).

Пересечения: {(NS, WN), (NE, EW), (SW, ES)}



Для заданного варианта необходимо создать набор свойств (LTL), которые определяют корректность работы модели (например отсутствие варианта, когда движение разрешено всем), а так же описать саму модель на языке Promela.

## Реализация

### Основная идея

Основной идеей данной реализации является отождествление пересечений направлений движения с ресурсами, которые необходимо получить контролеру для разрешения безопасного проезда. Данный подход обладает целым рядом преимуществ, из которых можно выделить:

- 1) Контролеры, управляющие пересекающимися направлениями, не могут дать зеленый свет одновременно
- 2) «Дружественные» контроллеры имеют возможность пропускать поток независимо
- 3) Данный подход довольно просто в реализации

### Модель на языке Promela

Данная модель была построена на основе следующих постулатов:

- 1) Каждый контроллер светофора является отдельным процессом.
- 2) Сигнал светофора может быть двух видов: красный – зеленый
- 3) Состояние светофора описывается глобальными переменными
- 4) Датчики движения также являются глобальными переменными
- 5) Движение (трафик машин) генерируется внешним, по отношению к контроллерам, процессом случайным образом

Реализация:

```
/* Course work made by Anton Lukashin group 6084-12 */
/* Exercise 1,12,15 (WN,NS) (NE, EW) (SW, ES) */
/* Types of signals */
mtype = {Red, Green};
/* Lights signals */
mtype NS_L = Red;
mtype WN_L = Red;
mtype NE_L = Red;
mtype EW_L = Red;
mtype ES_L = Red;
mtype SW_L = Red;
/* Car traffic sensors */
bool NS_S = false;
bool WN_S = false;
bool NE_S = false;
bool EW_S = false;
bool ES_S = false;
bool SW_S = false;
#define pNS_S (NS_L==Green && WN_L==Green && SW_L==Green && EW_L==Green)
#define pWN_S (WN_L==Green && NS_L==Green && NE_L==Green && SW_L==Green && EW_L==Green)
#define pEW_S (EW_L==Green && NE_L==Green && WN_L==Green && NS_L==Green)
#define pNE_S (NE_L==Green && WN_L==Green && ES_L==Green && EW_L==Green)
#define pSW_S (SW_L==Green && ES_L==Green && WN_L==Green && NS_L==Green)
#define pES_S (SW_L==Green && ES_L==Green && WN_L==Green && NS_L==Green)
#define pNS_L (NS_S && (NS_L==Red))
#define qNS_L (NS_L==Green)
#define pWN_L (WN_S && (WN_L==Red))
#define qWN_L (WN_L==Green)
#define pEW_L (EW_S && (EW_L==Red))
#define qEW_L (EW_L==Green)
#define pNE_L (NE_S && (NE_L==Red))
#define qNE_L (NE_L==Green)
#define pSW_L (SW_S && (SW_L==Red))
#define qSW_L (SW_L==Green)
#define pES_L (ES_S && (ES_L==Red))
#define qES_L (ES_L==Green)
#define pNS_F ((NS_L==Green) && NS_S)
#define pWN_F ((WN_L==Green) && WN_S)
```

```

#define pEW_F ((EW_L==Green) && EW_S)
#define pNE_F ((NE_L==Green) && NE_S)
#define pSW_F ((SW_L==Green) && SW_S)
#define pES_F ((ES_L==Green) && ES_S)
/*Safety*/
ltl p0_1 {[ !pNS_S}
ltl p0_2 {[ !pWN_S}
ltl p0_3 {[ !pEW_S}
ltl p0_4 {[ !pNE_S}
ltl p0_5 {[ !pSW_S}
ltl p0_6 {[ !pES_S}
/*Liveness*/
ltl p1_1 {[ (!pNS_L || (<> qNS_L))}
ltl p1_2 {[ (pWN_L -> (<> qWN_L))}
ltl p1_3 {[ (pEW_L -> (<> qEW_L))}
ltl p1_4 {[ (pNE_L -> (<> qNE_L))}
ltl p1_5 {[ (pSW_L -> (<> qSW_L))}
ltl p1_6 {[ (pES_L -> (<> qES_L))}
/*Fairness*/
ltl p2_1 {[ <> !pNS_F -> ([ <> (!pNS_F) && (pNS_L -> (<> qNS_L)))}
ltl p2_2 {[ <> !pWN_F -> ([ <> (!pWN_F) && (pWN_L -> (<> qWN_L)))}
ltl p2_3 {[ <> !pEW_F -> ([ <> (!pEW_F) && (pEW_L -> (<> qEW_L)))}
ltl p2_4 {[ <> !pNE_F -> ([ <> (!pNE_F) && (pNE_L -> (<> qNE_L)))}
ltl p2_5 {[ <> !pES_F -> ([ <> (!pES_F) && (pES_L -> (<> qES_L)))}
ltl p2_6 {[ <> !pSW_F -> ([ <> (!pSW_F) && (pSW_L -> (<> qSW_L)))}
/* Synchronization channels */
chan NS_WN_EW = [1] of {bool};
chan NS_WN_SW = [1] of {bool};
chan NE_WN_EW = [1] of {bool};
chan NE_ES = [1] of {bool};
chan ES_SW = [1] of {bool};
init
{
    atomic{
        NS_WN_EW ! true;
        NS_WN_SW ! true;
        NE_WN_EW ! true;
        NE_ES ! true;
        ES_SW ! true;
    };
    atomic{
        run NS();
        run WN();
        run NE();
        run ES();
        run EW();
        run SW();
        run gen_t();
    };
}
/* Traffic generation process */
proctype gen_t ()
{
    accpet: do
        :: true->
            if
                :: (NS_L==Green || !NS_S) -> NS_S = !NS_S;
                :: else -> skip;
            fi
        :: true->
            if
                :: (WN_L==Green || !WN_S) -> WN_S = !WN_S;
                :: else -> skip;
            fi
        :: true->
            if
                :: (NE_L==Green || !NE_S) -> NE_S = !NE_S;
                :: else -> skip;
            fi
        :: true->
            if
                :: (EW_L==Green || !EW_S) -> EW_S = !EW_S;
                :: else -> skip;
            fi
        :: true->
            if
                :: (ES_L==Green || !ES_S) -> ES_S = !ES_S;
                :: else -> skip;
            fi
        :: true->

```

```

        if
            :: (SW_L==Green || !SW_S) -> SW_S = !SW_S;
            :: else -> skip;
        fi
    od;
}
/* NS controller */
proctype NS ()
{
    end:
    do
        :: NS_S ->
            /* Wait for resources */
            NS_WN_EW ? true; NS_WN_SW ? true;
            NS_L = Green;
            if
                /* Wait for end of car queue */
                :: !NS_S -> skip;
            fi;
            NS_L = Red;
            NS_WN_EW ! true; NS_WN_SW ! true;
        od;
    }
/* WN controller */
proctype WN ()
{
    end:
    do
        :: WN_S ->
            /* Wait for resources */
            NS_WN_EW ? true; NS_WN_SW ? true; NE_WN_EW ? true;
            WN_L = Green;
            if
                /* Wait for end of car queue */
                :: !WN_S -> skip;
            fi;
            WN_L = Red;
            NS_WN_EW ! true; NS_WN_SW ! true; NE_WN_EW ! true;
        od;
    }
/* NE controller */
proctype NE ()
{
    end:
    do
        :: NE_S ->
            /* Wait for resources */
            NE_WN_EW ? true; NE_ES ? true;
            NE_L = Green;
            if
                /* Wait for end of car queue */
                :: !NE_S -> skip;
            fi;
            NE_L = Red;
            NE_WN_EW ! true; NE_ES ! true;
        od;
    }
/* NE controller */
proctype EW ()
{
    end:
    do
        :: EW_S ->
            /* Wait for resources */
            NS_WN_EW ? true; NE_WN_EW ? true;
            EW_L = Green;
            if
                /* Wait for end of car queue */
                :: !EW_S -> skip;
            fi;
            EW_L = Red;
            NS_WN_EW ! true; NE_WN_EW ! true;
        od;
    }
/* ES controller */
proctype ES ()
{
    end:
    do
        :: ES_S ->

```

```

/* Wait for resources */
ES_SW ? true; NE_ES ? true;
ES_L = Green;
if
/* Wait for end of car queue */
:: !ES_S -> skip;
fi;
ES_L = Red;
ES_SW ! true; NE_ES ! true;
od;
}
/* SW controller */
proctype SW ()
{
    end:
    do
        :: SW_S ->
            /* Wait for resources */
            NS_WN_SW ? true; ES_SW ? true;
            SW_L = Green;
            if
                /* Wait for end of car queue */
                :: !SW_S -> skip;
            fi;
            SW_L = Red;
            NS_WN_SW ! true; ES_SW ! true;
        od;
    }
}

```

## LTL правила

Данные правила на языке темпоральной логики оперируют двумя базовыми понятиями:

- 1) **G** ( $\square$  – в системе Spin) – описывает свойство, которое должно выполняться всегда
- 2) **F** ( $\diamond$  - в системе Spin) – описывает свойство, которое должно выполниться когда-то в будущем

Правила данной системы:

## Безопасность

Для данной модели правила безопасности звучат на естественном языке следующим образом: «Никогда не будет такой ситуации, что на данном направлении будет гореть зеленый свет, и на всех, пересекающих это направление дорогах, тоже будет зеленый»

- 1) **NS** -  $\text{ltl } p0\_1 \{ [] !pNS\_S \}$   
 $pNS\_S \text{ (NS\_L==Green \&\& WN\_L==Green \&\& SW\_L==Green \&\& EW\_L==Green)}$
- 2) **WN** -  $\text{ltl } p0\_2 \{ [] !pWN\_S \}$   
 $pWN\_S \text{ (WN\_L==Green \&\& NS\_L==Green \&\& NE\_L==Green \&\& SW\_L==Green \&\& EW\_L==Green)}$
- 3) **NE** -  $\text{ltl } p0\_3 \{ [] !pEW\_S \}$   
 $pEW\_S \text{ (EW\_L==Green \&\& NE\_L==Green \&\& WN\_L==Green \&\& NS\_L==Green)}$
- 4) **EW** -  $\text{ltl } p0\_4 \{ [] !pNE\_S \}$   
 $pNE\_S \text{ (NE\_L==Green \&\& WN\_L==Green \&\& ES\_L==Green \&\& EW\_L==Green)}$
- 5) **SW** -  $\text{ltl } p0\_5 \{ [] !pSW\_S \}$   
 $pSW\_S \text{ (SW\_L==Green \&\& ES\_L==Green \&\& WN\_L==Green \&\& NS\_L==Green)}$
- 6) **ES** -  $\text{ltl } p0\_6 \{ [] !pES\_S \}$   
 $pES\_S \text{ (SW\_L==Green \&\& ES\_L==Green \&\& WN\_L==Green \&\& NS\_L==Green)}$

## Живость и справедливость

Для данной модели правила **живости (liveness)** звучат на естественном языке следующим образом: «Всегда выполняется, если светофор горит красным и датчик показывает наличие машин, следовательно в будущем данный светофор будет гореть зеленым». А правила **справедливости (fairness)** - «Невозможна такая ситуация, что в конкретном направлении движется непрерывный поток машин(т.е. светофор должен неопределенно часто менять свой сигнал на запрещающий)».

Доказывать свойства fairness и liveness отдельно для данной модели не представляется возможным из-за того, что система Spin обладает «случайным» планировщиком процессов, который не гарантирует выполнения всех процессов. Таким образом, может сложиться ситуация, при которой один или несколько процессов будут выполняться всегда, в то время как все остальные процессы будут простаивать. Ясно, что в такой ситуации свойства fairness и liveness для простаивающих процессов выполняться не будут. Таким образом, свойства fairness и liveness надо доказывать вместе. На естественном языке объединенное свойство будет звучать следующим образом: «При наличии ожидающих автомобилей на каком-либо направлении ему обязательно представится возможность проехать (возможно, через какое-то время), при ограничении, что в каждом направлении не движется непрерывный поток автомобилей».

- 1) **NS** - { [] <> !pNS\_F -> ( [] <> (!pNS\_F) && (pNS\_L -> (<> qNS\_L))) }  
pNS\_F ((NS\_L==Green) && NS\_S)  
pNS\_L (NS\_S && (NS\_L==Red))  
qNS\_L (NS\_L==Green)
- 2) **WN** - { [] <> !pWN\_F -> ( [] <> (!pWN\_F) && (pWN\_L -> (<> qWN\_L))) }  
pWN\_F ((WN\_L==Green) && WN\_S)  
pWN\_L (WN\_S && (WN\_L==Red))  
qWN\_L (WN\_L==Green)
- 3) **NE** - { [] <> !pEW\_F -> ( [] <> (!pEW\_F) && (pEW\_L -> (<> qEW\_L))) }  
pNE\_F ((NE\_L==Green) && NE\_S)  
pNE\_L (NE\_S && (NE\_L==Red))  
qNE\_L (NE\_L==Green)
- 4) **EW** - { [] <> !pNE\_F -> ( [] <> (!pNE\_F) && (pNE\_L -> (<> qNE\_L))) }  
pEW\_F ((EW\_L==Green) && EW\_S)  
pEW\_L (EW\_S && (EW\_L==Red))  
qEW\_L (EW\_L==Green)
- 5) **SW** - { [] <> !pES\_F -> ( [] <> (!pES\_F) && (pES\_L -> (<> qES\_L))) }  
pSW\_F ((SW\_L==Green) && SW\_S)  
pSW\_L (SW\_S && (SW\_L==Red))  
qSW\_L (SW\_L==Green)
- 6) **ES** - { [] <> !pSW\_F -> ( [] <> (!pSW\_F) && (pSW\_L -> (<> qSW\_L))) }  
pES\_F ((ES\_L==Green) && ES\_S)  
pES\_L (ES\_S && (ES\_L==Red))  
qES\_L (ES\_L==Green)



## Результаты моделирования

Моделирование показало, что все проверки на безопасность и живучесть при условии справедливости система проходит успешно. Результаты моделирования для одной из проверок(живучесть при условии справедливости для направления NS) приведены ниже:

```
gcc -DMEMLIM=4500 -O2 -DXUSAFE -w -o pan pan.c
./pan -m100000000 -a -N p2_1
Pid: 2161
Depth= 406709 States= 1e+06 Transitions= 2.9e+06 Memory= 723.972 t=1.48 R=7e+05
Depth= 1000496 States= 2e+06 Transitions= 6.28e+06 Memory= 800.925 t=3.29 R=6e+05
Depth= 1533149 States= 3e+06 Transitions= 9.9e+06 Memory= 880.124 t=5.27 R=6e+05
Depth= 2070939 States= 4e+06 Transitions= 1.36e+07 Memory= 963.327 t=7.35 R=5e+05
Depth= 2528871 States= 5e+06 Transitions= 1.74e+07 Memory= 1046.628 t=9.53 R=5e+05
Depth= 2941845 States= 6e+06 Transitions= 2.12e+07 Memory= 1129.636 t=11.7 R=5e+05
Depth= 3341183 States= 7e+06 Transitions= 2.51e+07 Memory= 1215.671 t=14 R=5e+05
Depth= 3707041 States= 8e+06 Transitions= 2.89e+07 Memory= 1302.097 t=16.4 R=5e+05
Depth= 4068597 States= 9e+06 Transitions= 3.28e+07 Memory= 1389.499 t=18.8 R=5e+05
Depth= 4413393 States= 1e+07 Transitions= 3.66e+07 Memory= 1475.925 t=21.2 R=5e+05
Depth= 4776589 States= 1.1e+07 Transitions= 4.04e+07 Memory= 1562.741 t=23.6 R=5e+05
Depth= 5128173 States= 1.2e+07 Transitions= 4.41e+07 Memory= 1648.484 t=26.1 R=5e+05
Depth= 5442981 States= 1.3e+07 Transitions= 4.79e+07 Memory= 1734.812 t=28.5 R=5e+05
Depth= 5807473 States= 1.4e+07 Transitions= 5.17e+07 Memory= 1821.921 t=31 R=5e+05
Depth= 6116971 States= 1.5e+07 Transitions= 5.55e+07 Memory= 1909.616 t=33.6 R=4e+05
Depth= 6414843 States= 1.6e+07 Transitions= 5.93e+07 Memory= 1997.898 t=36.2 R=4e+05
Depth= 6642915 States= 1.7e+07 Transitions= 6.31e+07 Memory= 2086.667 t=38.9 R=4e+05
Depth= 6670587 States= 1.8e+07 Transitions= 7.6e+07 Memory= 2174.655 t=48.9 R=4e+05
Depth= 6670587 States= 1.9e+07 Transitions= 8.25e+07 Memory= 2266.257 t=53.4 R=4e+05
Depth= 6670587 States= 2e+07 Transitions= 8.96e+07 Memory= 2357.859 t=58.3 R=3e+05
Depth= 6670587 States= 2.1e+07 Transitions= 9.68e+07 Memory= 2449.558 t=63.5 R=3e+05
Depth= 6670587 States= 2.2e+07 Transitions= 1.04e+08 Memory= 2541.159 t=68.8 R=3e+05
Depth= 6670587 States= 2.3e+07 Transitions= 1.11e+08 Memory= 2632.761 t=74.4 R=3e+05
Depth= 6670587 States= 2.4e+07 Transitions= 1.19e+08 Memory= 2724.362 t=79.9 R=3e+05
Depth= 6670587 States= 2.5e+07 Transitions= 1.26e+08 Memory= 2815.964 t=85.4 R=3e+05
Depth= 6670587 States= 2.6e+07 Transitions= 1.33e+08 Memory= 2907.566 t=90.9 R=3e+05
Depth= 6670587 States= 2.7e+07 Transitions= 1.41e+08 Memory= 2999.167 t=96.6 R=3e+05
Depth= 6670587 States= 2.8e+07 Transitions= 1.48e+08 Memory= 3090.769 t=102 R=3e+05
Depth= 6670587 States= 2.9e+07 Transitions= 1.55e+08 Memory= 3182.370 t=108 R=3e+05
Depth= 6670587 States= 3e+07 Transitions= 1.63e+08 Memory= 3273.972 t=114 R=3e+05
Depth= 6670587 States= 3.1e+07 Transitions= 1.7e+08 Memory= 3365.573 t=120 R=3e+05
Depth= 6670587 States= 3.2e+07 Transitions= 1.77e+08 Memory= 3457.273 t=126 R=3e+05
(Spin Version 6.2.5 -- 3 May 2013)
+ Partial Order Reduction
Full statespace search for:
    never claim + (p2_1)
    assertion violations + (if within scope of claim)
    acceptance cycles + (fairness disabled)
    invalid end states - (disabled by never claim)
State-vector 148 byte, depth reached 6670587, errors: 0
31367944 states, stored (3.28579e+07 visited)
1.594091e+08 states, matched
1.9226703e+08 transitions (= visited+matched)
26 atomic steps
hash conflicts: 1.014425e+08 (resolved)
Stats on memory usage (in Megabytes):
5265.005 equivalent memory usage for states (stored*(State-vector + overhead))
2875.560 actual memory usage for states (compression: 54.62%)
state-vector as stored = 68 byte + 28 byte overhead
128.000 memory used for hash table (-w24)
534.058 memory used for DFS stack (-m100000000)
1.829 memory lost to fragmentation
3535.788 total actual memory usage

pan: elapsed time 139 seconds
No errors found -- did you verify all claims?
```

## Выводы

В рамках данной курсовой работы были исследованы подходы к верификации распределенных программ. Для модельной задачи были определены LTL формулы, описывающие корректное поведение системы. Далее модель была описана на языке Promela в системе Spin. После чего была проведена верификация с учетом описанных в ответе LTL формул. По результатам верификации можно судить о корректности модели.

Таким образом можно говорить об успешном выполнении задания и освоении инструментов верификации. Данные знания являются актуальными и могут применяться в большом количестве практик современного программирования.

## Список литературы

- 1) Ю.Г. Карпов, И.В. Шошмина Верификация распределенных систем – СПб.: Издательство Политехнического университета, 2011.
- 2) Thomas Wahl – Fairness and Liveness URL:  
<http://www.ccs.neu.edu/home/wahl/Publications/fairness.pdf>
- 3) Concurrent programming lab2 URL:  
[http://www2.compute.dtu.dk/courses/02158/sol\\_cplab2.html](http://www2.compute.dtu.dk/courses/02158/sol_cplab2.html)
- 4) Andrew Ireland - Distributed Systems Programming (F21DS1) SPIN: Formal Analysis I URL: <http://www.macs.hw.ac.uk/~air/dsp-spin/lectures/lec-6-spin-2.pdf>
- 5) AG-Wehrheim – Verification with SPIN URL:  
[http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Wehrheim/Lehre/SS09/Model\\_Checking/Slides/11May09.pdf](http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Wehrheim/Lehre/SS09/Model_Checking/Slides/11May09.pdf)