

**МИНОБРНАУКИ РОССИИ**  
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**  
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**  
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**  
**Кафедра математического обеспечения и применения ЭВМ**

**Курсовая работа**  
**по дисциплине «Верификация распределённых алгоритмов»**  
**Тема: Контроллер светофоров**

Студент гр. 0303

Болкунов В.О.

Преподаватель

Шошмина И.В.

Санкт-Петербург

2025

## **ЗАДАНИЕ НА КУРСОВУЮ РАБОТУ**

Студент Болкунов В.О.

Группа 0303

Тема работы: контроллер светофоров

Исходные данные:

Разработать модель контроллера светофора, управляющего движением транспорта на сложном перекрёстке.

Содержание пояснительной записки: содержание, введение, ход работы, заключение, список использованных источников.

Предполагаемый объем пояснительной записки:

Не менее 16 страниц.

Дата выдачи задания: 01.04.2025

Дата сдачи реферата: 28.05.2025

Дата защиты реферата: 28.05.2025

Студент

Болкунов В.О.

Преподаватель

Шошмина И.В.

## СОДЕРЖАНИЕ

<b>Ход работы.....</b>	<b>5</b>
Постановка задачи.....	5
Состояние модели.....	5
Процессы.....	6
<b>Выводы.....</b>	<b>11</b>
СПИСОК ЛИТЕРАТУРЫ.....	12
ПРИЛОЖЕНИЕ А   ИСХОДНЫЙ КОД.....	13

## **Введение**

Целью работы является проектирование и разработка модели системы перекрёстка. Система моделирует поток автомобилей, каждое направление движений регулируется своим контроллером светофора. Контроллеры, представленные параллельными процессами, осуществляют борьбу за ресурсы - области перекрёстка, в которых один и более направлений движений имеют пересечение. Разрабатываемая модель должна отвечать требованиям безопасности, живости и справедливости.

## Ход работы

### Постановка задачи.

Вариант 2, направления: NS, EW, SD, WN, WD, DN.

Схема перекрёстка с указанными направлениями и их пересечениями представлена на рисунке 1.

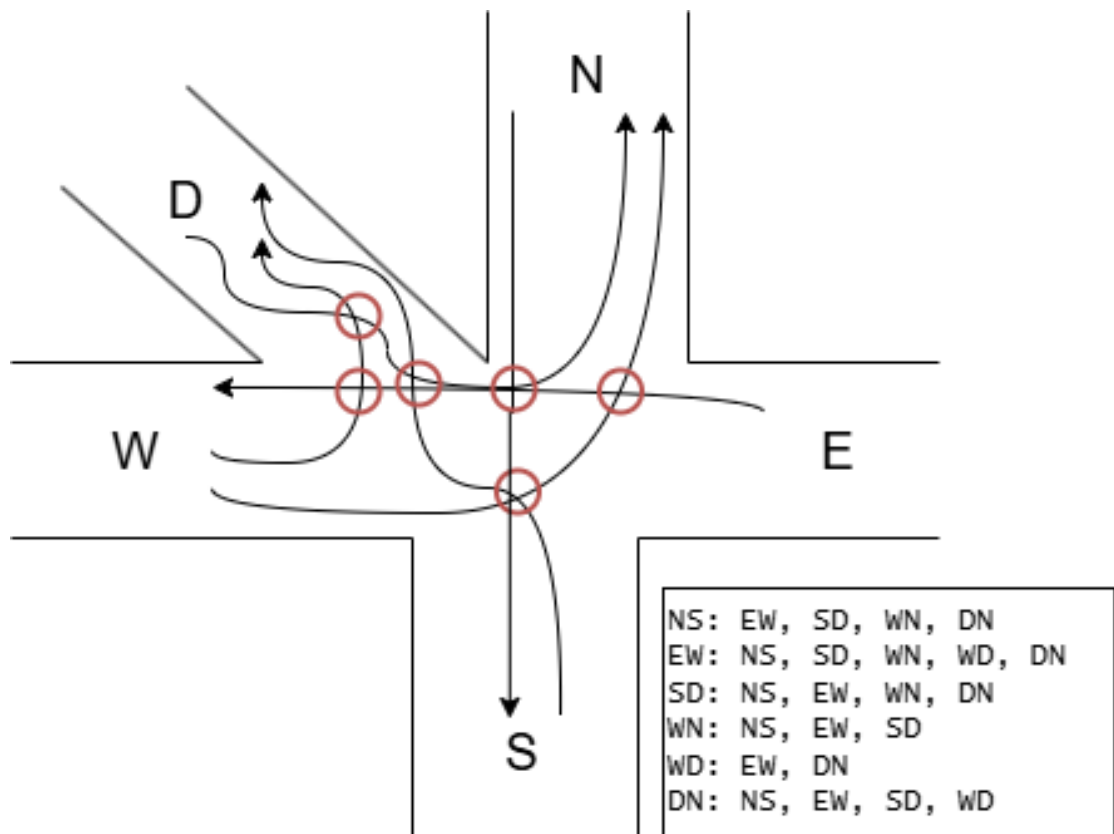


Рисунок 1 - Схема перекрёстка

### Состояние модели.

Для описания состояния контроллера направления реализована структура *TLight*.

```
typedef TLight {  
    int color;  
    bool sense;  
    bool req;  
    bool lock;  
}
```

Она содержит цвет *color* светофора, флаг наличия транспорта на направлении *sense*, запрос на пропуск *req* и флаг блокировки *lock*.

### Процессы.

Для моделирования дорожного движения реализован процесс *Traffic* и ещё 6 процессов для каждого направления.

Процесс трафика недетерминированно активирует датчики и устанавливает запросы на смену цвета светофора.

Процессы направления проверяют запросы на проезд машин в соответствующем направлении, а также проверяют наличие блокировки.

В случае отсутствия блокировки, производится атомарная проверка на наличие блокировок на конфликтующих направлениях; и, в случае их отсутствия, устанавливается блокировка и активирует зеленый сигнал светофора.

Следующим условием является наличие блокировки и отсутствие запроса на проезд. При его выполнении выключается сигнал светофора, а затем снимается блокировка.

Следующее условие необходимо для проверки наличия машин на направлении, при условии, что есть запрос на проезд. Если машины закончились, запрос на проезд снимается.

### Верификация.

Верификация модели осуществлена путём составления LTL формул для каждого направления по условиям безопасности, живости и справедливости.

Условие безопасности: «Никогда не будет ситуации, в которой при разрешающем сигнале одного светофора будут даны разрешающие сигналы светофорами, находящимися на конфликтующих направлениях.»

Пример для WD:

```
[ ] !((WD.color == GREEN) && (EW.color == GREEN || DN.color == GREEN))
```

Условие живости: «Всегда, если на направлении окажутся машины, в будущем светофор даст разрешающий сигнал»

Пример для NS:

```
[ ] ( (NS.sense && (NS.color == RED)) -> <> (NS.color == GREEN) )
```

Условие справедливости: «Всегда в будущем либо на направлении кончатся машины, либо выключится светофор.»

Пример для DN:

```
[ ] <> !((DN.color == GREEN) && DN.sense)
```

Для сборки модели использовались следующие параметры компилятора gcc: *-DMEMLIM=1024 -O2 -w*.

Для запуска модели использовались: *-m1000000 -a -n -c1*.

Примеры запуска верификатора для проверок условий безопасности, живости и справедливости представлены на рисунках 2, 3 и 4 соответственно.

```
ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    ПОРТЫ

PS D:\vl\лэти\Верификация> .\model.exe -m1000000 -a -n -c1 -N safetyEW
pan: ltl formula safetyEW
Depth=  18195 States=  1e+06 Transitions= 4.06e+06 Memory=  288.437 t=  0.935 R=  1e+06
Depth=  35551 States=  2e+06 Transitions= 8.63e+06 Memory=  395.273 t=  2.01 R=  1e+06
Depth=  41893 States=  3e+06 Transitions= 1.38e+07 Memory=  502.109 t=  3.23 R=  9e+05

(Spin Version 6.5.1 -- 31 July 2020)
+ Partial Order Reduction

Full statespace search for:
    never claim          + (safetyEW)
    assertion violations + (if within scope of claim)
    acceptance cycles   + (fairness disabled)
    invalid end states   - (disabled by never claim)

State-vector 108 byte, depth reached 41893, errors: 0
    3225686 states, stored
    11427595 states, matched
    14653281 transitions (= stored+matched)
        0 atomic steps
hash conflicts: 718496 (resolved)

Stats on memory usage (in Megabytes):
    393.760    equivalent memory usage for states (stored*(State-vector + overhead))
    344.993    actual memory usage for states (compression: 87.61%)
               state-vector as stored = 92 byte + 20 byte overhead
    128.000    memory used for hash table (-w24)
    53.406    memory used for DFS stack (-m1000000)
    526.230    total actual memory usage

pan: elapsed time 3.48 seconds
pan: rate 926921.26 states/second
```

Рисунок 2 - Верификация условия безопасности для направления EW



```
ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    ПОРТЫ

● PS D:\vl\лэти\Верификация> .\model.exe -m1000000 -a -n -c1 -N livenessSD
pan: ltl formula livenessSD
Depth=   2368 States=   1e+06 Transitions= 4.29e+06 Memory=   236.875 t=   0.796 R=   1e+06
Depth=   2379 States=   2e+06 Transitions= 9.32e+06 Memory=   290.293 t=   1.74 R=   1e+06
Depth=   7987 States=   3e+06 Transitions= 1.49e+07 Memory=   360.703 t=   2.91 R=   1e+06
Depth=  16503 States=   4e+06 Transitions= 2.05e+07 Memory=   433.945 t=   4.08 R=   1e+06
Depth=  22003 States=   5e+06 Transitions= 2.7e+07 Memory=   514.218 t=   5.58 R=   9e+05
Depth=  32587 States=   6e+06 Transitions= 3.28e+07 Memory=   596.933 t=   6.9 R=   9e+05
Depth=  41893 States=   7e+06 Transitions= 4e+07 Memory=   686.093 t=   8.57 R=   8e+05
Depth=  41893 States=   8e+06 Transitions= 4.7e+07 Memory=   773.984 t=  10.2 R=   8e+05

(Spin Version 6.5.1 -- 31 July 2020)
+ Partial Order Reduction

Full statespace search for:
    never claim          + (livenessSD)
    assertion violations + (if within scope of claim)
    acceptance cycles   + (fairness disabled)
    invalid end states   - (disabled by never claim)

State-vector 108 byte, depth reached 41893, errors: 0
  5681927 states, stored (8.13817e+06 visited)
  39356594 states, matched
  47494762 transitions (= visited+matched)
    0 atomic steps
hash conflicts: 3908359 (resolved)

Stats on memory usage (in Megabytes):
  693.595    equivalent memory usage for states (stored*(State-vector + overhead))
  607.577    actual memory usage for states (compression: 87.60%)
              state-vector as stored = 92 byte + 20 byte overhead
  128.000    memory used for hash table (-w24)
   53.406    memory used for DFS stack (-m1000000)
  788.730    total actual memory usage

pan: elapsed time 10.4 seconds
pan: rate 785537.45 states/second
```

Рисунок 3 - Верификация условия живости для направления SD

```
ПРОБЛЕМЫ    ВЫХОДНЫЕ ДАННЫЕ    КОНСОЛЬ ОТЛАДКИ    ТЕРМИНАЛ    ПОРТЫ

● PS D:\v1\лэти\Верификация> .\model.exe -m1000000 -a -n -c1 -N fairnessNS
pan: ltl formula fairnessNS
Depth= 16989 States= 1e+06 Transitions= 3.82e+06 Memory= 276.621 t= 0.863 R= 1e+06
Depth= 30623 States= 2e+06 Transitions= 8.3e+06 Memory= 376.914 t= 1.89 R= 1e+06
Depth= 41893 States= 3e+06 Transitions= 1.33e+07 Memory= 476.230 t= 3.05 R= 1e+06

(Spin Version 6.5.1 -- 31 July 2020)
+ Partial Order Reduction

Full statespace search for:
    never claim          + (fairnessNS)
    assertion violations + (if within scope of claim)
    acceptance cycles   + (fairness disabled)
    invalid end states   - (disabled by never claim)

State-vector 108 byte, depth reached 41893, errors: 0
 3508254 states, stored (3.79082e+06 visited)
 13567684 states, matched
 17358506 transitions (= visited+matched)
    0 atomic steps
hash conflicts: 880863 (resolved)

Stats on memory usage (in Megabytes):
 428.254    equivalent memory usage for states (stored*(State-vector + overhead))
 375.179    actual memory usage for states (compression: 87.61%)
            state-vector as stored = 92 byte + 20 byte overhead
 128.000    memory used for hash table (-w24)
  53.406    memory used for DFS stack (-m1000000)
 556.406    total actual memory usage

pan: elapsed time 4.06 seconds
pan: rate 933930.03 states/second
```

Рисунок 4 - Верификация условия справедливости для направления NS

### **Выводы.**

В результате выполнения работы была спроектирована модель системы сложного перекрёстка. Реализована программа на языке promela, описывающая данную модель. С помощью параллельных процессов смоделирован трафик и контроллеры светофоров заданных направлений.

Разработаны условия верификации спроектированной модели по условиям безопасности, живости и справедливости. Разработанная модель была успешно протестирована по заданным условиям для каждого направления.

## СПИСОК ЛИТЕРАТУРЫ

1. Карпов Ю. Г., Шошмина И. В. Верификация распределенных систем: Санкт-Петербургский государственный политехнический университет, 2011
2. Документация Spin // <https://spinroot.com/spin/Man/Spin.html> (дата обращения: 20.05.2025).
3. Документация Promela // <https://spinroot.com/spin/Man/promela.html> (дата обращения: 20.05.2025).

## ПРИЛОЖЕНИЕ А

### ИСХОДНЫЙ КОД

```
/* Структура контроллера одного направления */
typedef TLight {
    int color;
    bool sense;
    bool req;
    bool lock;
}

mtype = { GREEN, RED }

TLight NS, EW, SD, WN, WD, DN;

/* Условия безопасности */
ltl safetyNS { [] !((NS.color == GREEN) && (EW.color == GREEN || SD.color == GREEN
|| WN.color == GREEN || DN.color == GREEN)) }
ltl safetyEW { [] !((EW.color == GREEN) && (NS.color == GREEN || SD.color == GREEN
|| WN.color == GREEN || WD.color == GREEN || DN.color == GREEN)) }
ltl safetySD { [] !((SD.color == GREEN) && (NS.color == GREEN || EW.color == GREEN
|| WN.color == GREEN || DN.color == GREEN)) }
ltl safetyWN { [] !((WN.color == GREEN) && (NS.color == GREEN || SD.color == GREEN
|| EW.color == GREEN)) }
ltl safetyWD { [] !((WD.color == GREEN) && (EW.color == GREEN || DN.color ==
GREEN)) }
ltl safetyDN { [] !((DN.color == GREEN) && (NS.color == GREEN || EW.color == GREEN
|| SD.color == GREEN || WD.color == GREEN)) }

/* Условия живости */
ltl livenessNS { [] ((NS.sense && (NS.color == RED)) -> <> (NS.color == GREEN)) }
ltl livenessEW { [] ((EW.sense && (EW.color == RED)) -> <> (EW.color == GREEN)) }
ltl livenessSD { [] ((SD.sense && (SD.color == RED)) -> <> (SD.color == GREEN)) }
ltl livenessWN { [] ((WN.sense && (WN.color == RED)) -> <> (WN.color == GREEN)) }
ltl livenessWD { [] ((WD.sense && (WD.color == RED)) -> <> (WD.color == GREEN)) }
ltl livenessDN { [] ((DN.sense && (DN.color == RED)) -> <> (DN.color == GREEN)) }

/* Условия справедливости */
ltl fairnessNS { [] <> !((NS.color == GREEN) && NS.sense) }
ltl fairnessEW { [] <> !((EW.color == GREEN) && EW.sense) }
ltl fairnessSD { [] <> !((SD.color == GREEN) && SD.sense) }
ltl fairnessWN { [] <> !((WN.color == GREEN) && WN.sense) }
ltl fairnessWD { [] <> !((WD.color == GREEN) && WD.sense) }
ltl fairnessDN { [] <> !((DN.color == GREEN) && DN.sense) }

/* Моделирование трафика */
active proctype Traffic() {
    do
        :: atomic {
            /* Если нигде нет машин - устанавливаем везде */
            if
                :: (! (
                    NS.sense || EW.sense || SD.sense ||
                    WN.sense || WD.sense || DN.sense
                )) ->
                    NS.sense = true;
                    EW.sense = true;
                    SD.sense = true;
                    WN.sense = true;
                    WD.sense = true;
                    DN.sense = true;
            fi
        }
    od
}
```

```

}

/* Для каждого направления */
/* - Если есть машины и нет запроса - устанавливаем запрос */
/* - Если есть машины и горит зелёный - убираем машины (считаем что проехали)
*/

:: (NS.sense && !NS.req) ->
    NS.req = true;
:: (NS.sense && NS.color == GREEN) ->
    NS.sense = false;

:: (EW.sense && !EW.req) ->
    EW.req = true;
:: (EW.sense && EW.color == GREEN) ->
    EW.sense = false;

:: (SD.sense && !SD.req) ->
    SD.req = true;
:: (SD.sense && SD.color == GREEN) ->
    SD.sense = false;

:: (WN.sense && !WN.req) ->
    WN.req = true;
:: (WN.sense && WN.color == GREEN) ->
    WN.sense = false;

:: (WD.sense && !WD.req) ->
    WD.req = true;
:: (WD.sense && WD.color == GREEN) ->
    WD.sense = false;

:: (DN.sense && !DN.req) ->
    DN.req = true;
:: (DN.sense && DN.color == GREEN) ->
    DN.sense = false;
od
}

/* Процесс для каждого направления (светофора) */
active proctype NSproc() {
    do
        /* Если есть запрос и нет блокировки переходим в atomic блок */
        :: (NS.req && !NS.lock) ->
            atomic {
                /* Если нигде больше нет блокировок, устанавливаем блокировку и цвет
                зелёный */
                if
                    :: (!EW.lock && !SD.lock && !WN.lock && !DN.lock) ->
                        NS.lock = true;
                        NS.color = GREEN;
                fi
            }
        /* Если есть блокировка и нет запроса - красный */
        :: (NS.lock && !NS.req) ->
            NS.color = RED;
            NS.lock = false;
        /* Если нет машин и есть запрос - убираем запрос */
        :: (!NS.sense && NS.req) ->
            NS.req = false;
    od
}

```

```

active proctype EWproc() {
    do
        :: (EW.req && !EW.lock) ->
            atomic {
                if
                    :: (!NS.lock && !SD.lock && !WN.lock && !WD.lock && !DN.lock) ->
                        EW.lock = true;
                        EW.color = GREEN;
                    fi
                }
            :: (EW.lock && !EW.req) ->
                EW.color = RED;
                EW.lock = false;
            :: (!EW.sense && EW.req) ->
                EW.req = false;
        od
    }

active proctype SDproc() {
    do
        :: (SD.req && !SD.lock) ->
            atomic {
                if
                    :: (!NS.lock && !EW.lock && !WN.lock && !DN.lock) ->
                        SD.lock = true;
                        SD.color = GREEN;
                    fi
                }
            :: (SD.lock && !SD.req) ->
                SD.color = RED;
                SD.lock = false;
            :: (!SD.sense && SD.req) ->
                SD.req = false;
        od
    }

active proctype WNproc() {
    do
        :: (WN.req && !WN.lock) ->
            atomic {
                if
                    :: (!NS.lock && !EW.lock && !SD.lock) ->
                        WN.lock = true;
                        WN.color = GREEN;
                    fi
                }
            :: (WN.lock && !WN.req) ->
                WN.color = RED;
                WN.lock = false;
            :: (!WN.sense && WN.req) ->
                WN.req = false;
        od
    }

active proctype WDproc() {
    do
        :: (WD.req && !WD.lock) ->
            atomic {
                if
                    :: (!EW.lock && !DN.lock) ->
                        WD.lock = true;
                        WD.color = GREEN;
                    fi
                }
            }
    }

```

```

    }
    :: (WD.lock && !WD.req) ->
        WD.color = RED;
        WD.lock = false;
    :: (!WD.sense && WD.req) ->
        WD.req = false;
    od
}

active proctype DNproc() {
    do
        :: (DN.req && !DN.lock) ->
            atomic {
                if
                    :: (!NS.lock && !EW.lock && !SD.lock && !WD.lock) ->
                        DN.lock = true;
                        DN.color = GREEN;
                fi
            }
        :: (DN.lock && !DN.req) ->
            DN.color = RED;
            DN.lock = false;
        :: (!DN.sense && DN.req) ->
            DN.req = false;
    od
}

```