МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

КУРСОВАЯ РАБОТА

по дисциплине «Верификация распределенных алгоритмов»

Тема: Контроллер светофоров

Студент гр. 0303 _____ Калмак Д.А.

Преподаватель _____ Шошмина И.В.

Санкт-Петербург

2025

ЗАДАНИЕ
НА КУРСОВУЮ РАБОТУ

Студент Калмак Д.А.

Группа 0303

Тема работы: контроллер светофоров

Исходные данные:
Разработать модель контроллера светофора, управляющего движением автомобилей по сложному перекрестку с учетом: машины на каждом направлении движутся независимо, появление машины в каждом направлении регистрируется своим независимым датчиком движения, контроллер светофора в каждом направлении работает по алгоритму, данному в методичке для другого перекрестка.

Содержание пояснительной записки:
«Содержание», «Введение», «Выполнение работы», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:
Не менее 17 страниц.

Дата выдачи задания: 26.03.2025

Дата сдачи реферата: 28.05.2025

Дата защиты реферата: 28.05.2025

Студент _____ Калмак Д.А.

Преподаватель _____ Шошмина И.В.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель данной работы заключается в построении верифицированной относительно заданных свойств безопасности, живости и справедливости модели контроллера светофора, управляющего движением автомобилей по сложному перекрестку. Движение машин на разных направлениях происходит независимо, что разрешает движение нескольких непересекающихся автомобильных потоков. Появление машин фиксируется отдельным контроллером светофора, или направления, причем процессы параллельные, поэтому между контроллерами должна возникнуть борьба за ресурсы. Вместе с контроллерами светофоров требуется определить контроллер внешней среды.

1. ВЫПОЛНЕНИЕ РАБОТЫ

1.1 Заданный перекресток

Сложный перекресток состоит из семи направлений движения автомобилей: NS, SW, ED, WN, WD, DN, DE – соответствующих варианту 6. Каждое направление имеет пересечения с другими направлениями:

- NS: SW, ED, WN, WD, DE, DN – шесть конфликтов.
- SW: NS, WN, WD – три конфликта.
- ED: NS, WN, DN – три конфликта.
- WN: NS, SW, ED, DE – четыре конфликта.
- WD: NS, SW, DN, DE – четыре конфликта.
- DN: NS, ED, WD – три конфликта.
- DE: NS, WN, WD – три конфликта.

Перекресток с направлениями, пересечения которых выделены черными кругами, представлен на рис. 1.

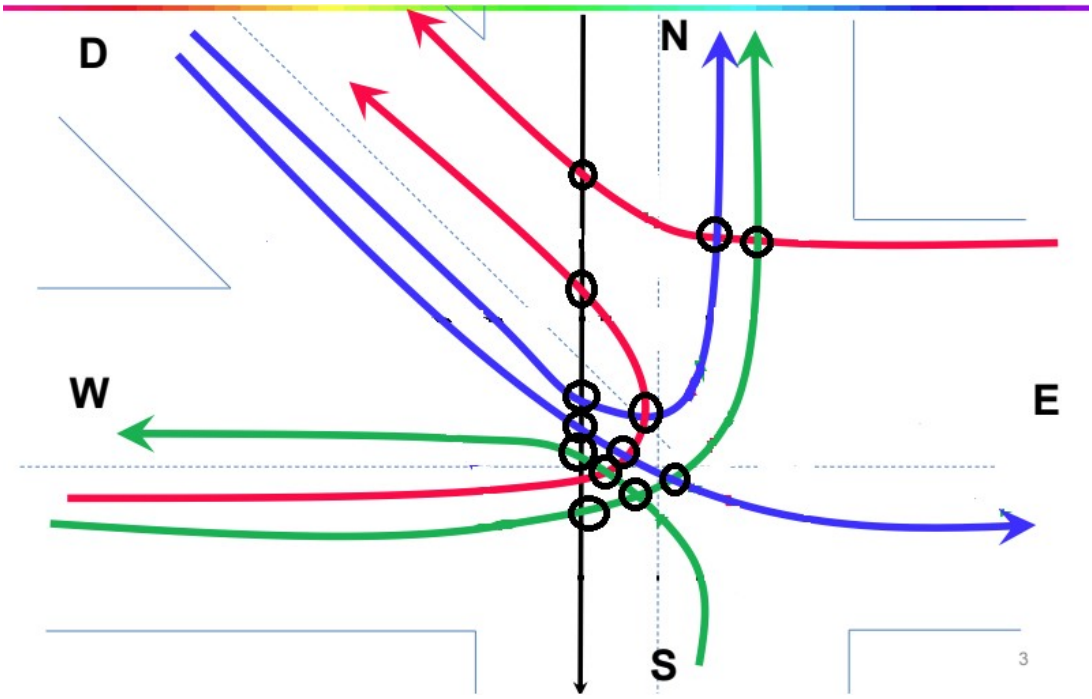

Рисунок 1 – Перекресток с пересечениями направлений

Для оптимизации программы было убрано направление DE. Резервное направление WN для оптимизации не понадобилось.

1.2 Решение задачи

Для контроллера светофоров требуются две глобальные переменные и массив каналов (мьютексов) для взаимного исключения направлений, конфликтующих на перекрестке. У каждого направления свой светофор, поэтому каждое направление имеет свои глобальные переменные. <направление> соответствует NS, SW, ED, WN, WD, DN. int <направление>_LIGHT – содержит сигнал светофора соответствующего направления. Для наглядности создано перечисление с символическими именами: GREEN и RED, соответствующие цвету светофора. Массивы каналов состоят из каналов пересечений и каждый канал имеет размер с количество направлений. Канал пересечения представляет собой очередь с индексами направлений, за которую борются направления, чтобы по своему направлению провести автомобильный поток. Последняя глобальная переменная bool <направление>_SENSE сообщает о наличии автомобилей на соответствующем направлении.

Для поведения среды выделен отдельный процесс env_controller. Данный процесс регулирует автомобильный поток случайным образом: когда нет автомобилей на направлении, он на направлении назначает автомобили. Процесс представляет собой бесконечное создание внешней среды.

Контроллеры светофоров представляют собой процессы с именем направления. Каждый процесс имеет одинаковую структуру. Если обнаружены на направлении автомобили и сигнал светофора красный, то в канал, который соответствует пересечению, передается индекс текущего направления для постановки в очередь и ожидаем, когда он станет первый в очереди, такие действия происходят для каждого пересечения. Когда текущее направление стало во всех пересечениях первым, включаем на светофоре зеленый сигнал для проезда автомобилей. Если сигнал светофора зеленый, то считаем, что автомобили проехали и на направлении их больше нет, включаем красный сигнал светофора и убираем со всех каналов пересечений индекс текущего направления, тем самым исключая его из очереди. Процесс контроллера

светофора, или направления, предназначен для контроля сигнала светофора и борьбы за пересечения, мешающие проезду по этому направлению, а также сообщает, что на направлении автомобилей нет.

Разработанный код представлен в Приложении А.

1.3 Верификация модели

Каждое направление проверено на безопасность, живость и справедливость для верификации модели.

Например, для направления NS:

- Свойство безопасности:

G ¬ ((SW_LIGHT == GREEN || ED_LIGHT == GREEN || WN_LIGHT == GREEN || WD_LIGHT == GREEN || DN_LIGHT == GREEN) & (NS_LIGHT == GREEN))

Никогда не включится разрешающий сигнал светофора одновременно и на выбранном направлении, и на пересекающихся с ним направлениях, или никогда не будет разрешен проезд в пересекающихся направлениях.

Проверка свойства безопасности для направления NS представлена на рис. 2.

```
pan: ltl formula safety_NS
Depth= 1583573 States=    1e+06 Transitions=  2.8e+06 Memory=  1746.893 t=     1.95 R=   5e+05
Depth= 2880285 States=    2e+06 Transitions= 6.27e+06 Memory=  2121.014 t=     4.38 R=   5e+05
Depth= 4121393 States=    3e+06 Transitions= 9.95e+06 Memory=  2495.233 t=     6.98 R=   4e+05
Depth= 5409127 States=    4e+06 Transitions= 1.37e+07 Memory=  2869.354 t=     9.72 R=   4e+05
Depth= 6620061 States=    5e+06 Transitions= 1.77e+07 Memory=  3243.572 t=     12.8 R=   4e+05
Depth= 7616353 States=    6e+06 Transitions= 2.21e+07 Memory=  3617.694 t=     16.3 R=   4e+05
Depth= 8487677 States=    7e+06 Transitions= 2.66e+07 Memory=  3991.815 t=     20.1 R=   3e+05
Depth= 8930751 States=    8e+06 Transitions= 3.22e+07 Memory=  4366.033 t=     24.4 R=   3e+05
Depth= 9074119 States=    9e+06 Transitions= 3.78e+07 Memory=  4740.154 t=     28.6 R=   3e+05
Depth= 9140277 States=    1e+07 Transitions= 4.38e+07 Memory=  5114.373 t=     33.2 R=   3e+05

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

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

State-vector 384 byte, depth reached 9140277, errors: 0
 10179202 states, stored
 48503124 states, matched
 58682326 transitions (= stored+matched)
        6 atomic steps
hash conflicts:  10787972 (resolved)

Stats on memory usage (in Megabytes):
 3921.888       equivalent memory usage for states (stored*(State-vector + overhead))
    0.000       actual memory usage for states (less than 1k)
  128.000       memory used for hash table (-w24)
 5340.576       memory used for DFS stack (-m100000000)
    3.332       memory lost to fragmentation
 5181.365       total actual memory usage



pan: elapsed time 46.1 seconds
pan: rate 220639.47 states/second
```

Рисунок 2 – Проверка свойства безопасности направления NS

- Свойство живости:

G ((NS_SENSE & (NS_LIGHT == RED)) -> F (NS_LIGHT == GREEN))

Всегда, если на направлении есть автомобили и включен запрещающий сигнал светофора, в будущем включится разрешающий сигнал светофора, или при появлении машины ей всегда предоставится возможность проезда в нужном направлении (возможно, не сразу).

Проверка свойства живости для направления NS представлена на рис. 3.

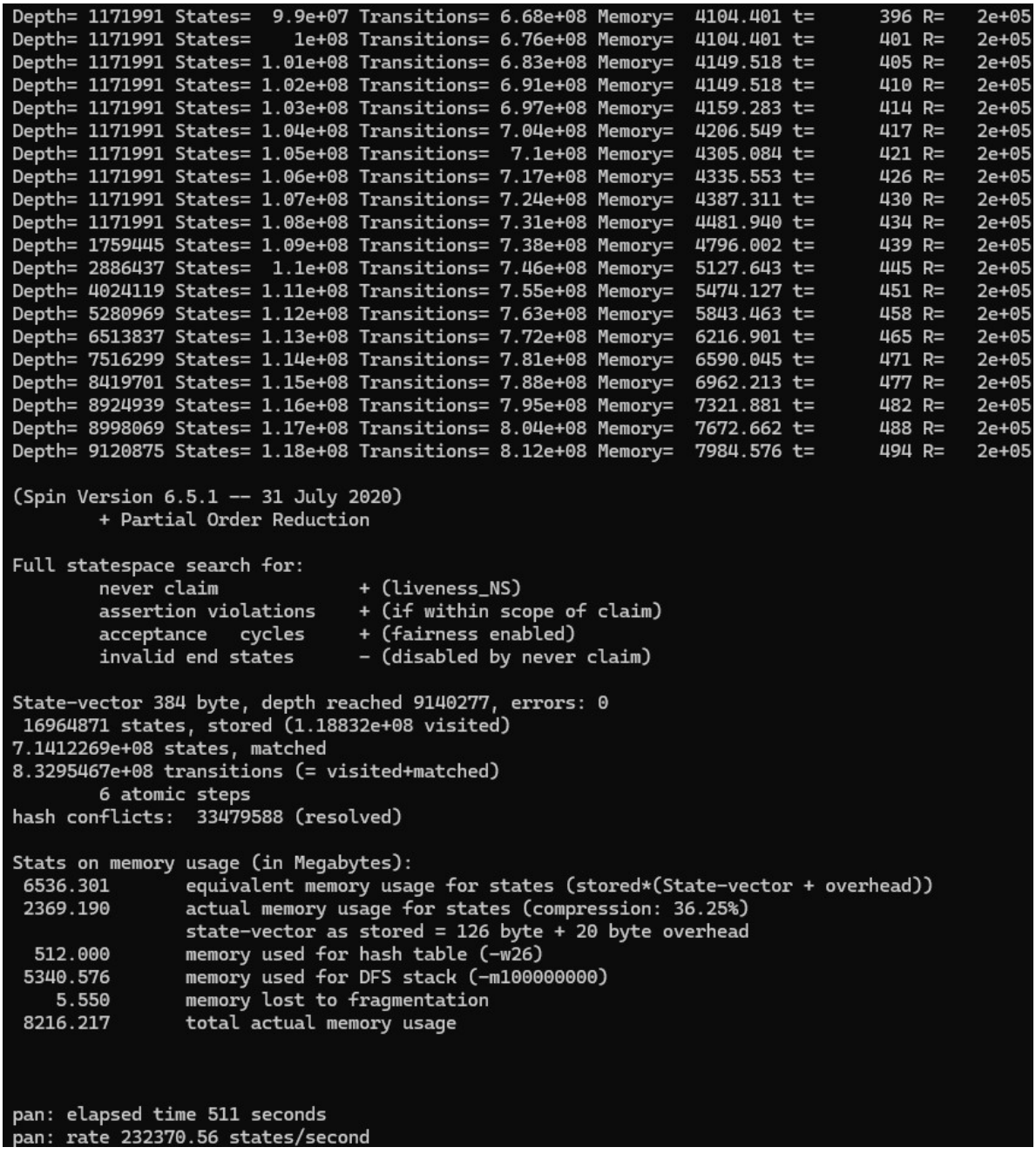

Рисунок 3 – Проверка свойства живости направления NS

- Свойство справедливости:

GF ¬ ((NS_LIGHT == GREEN) & NS_SENSE)

Всегда в будущем на направлении включится запрещающий сигнал светофора или закончатся автомобили, или в каждом направлении не движется непрерывный поток машин.

Проверка свойства справедливости для направления NS представлена на рис. 4.

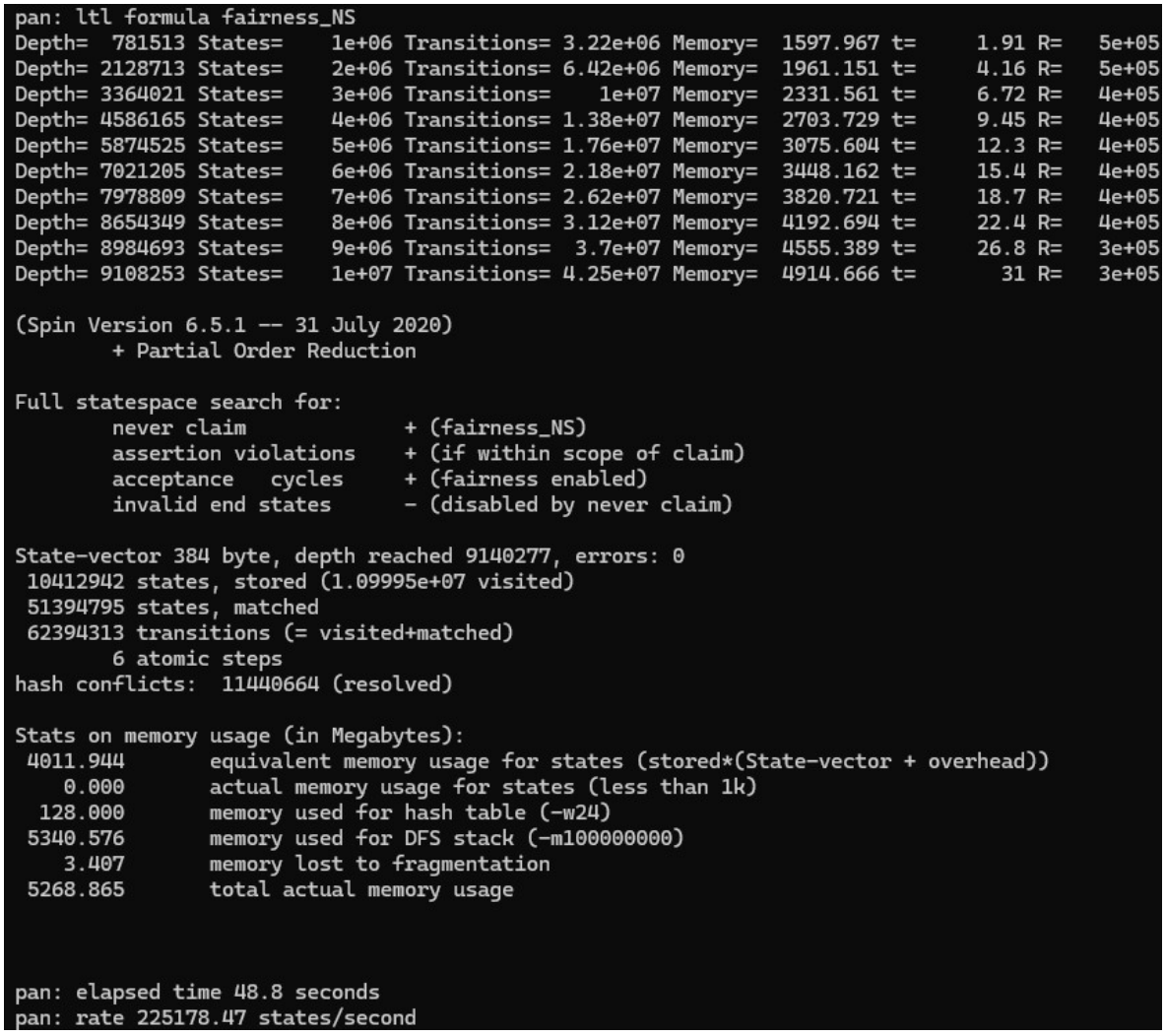

Рисунок 4 – Проверка свойства справедливости направления NS

На рис. 2-4 при сборке компилятором gcc использовались параметры: -O2 -DXUSAFE -DNFAIR=3 -w, при запуске модели: -m100000000 -a -f -n -c1. Верификация проходит также и при других параметрах.

ЗАКЛЮЧЕНИЕ

Таким образом, была разработана модель сложного перекрестка с использованием языка Promela и верификатора Spin. Модель включает разработанный контроллер светофоров, или направлений, и контроллер внешней среды, основанные на параллельных процессах. Были заданы свойства безопасности, живости и справедливости. Модель была успешно верифицирована для каждого направления по каждому свойству.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Ю.Г. Карпов, И.В. Шошмина Верификация распределенных систем: Издательство Политехнического университета, 2011.

2. Документация Promela // Promela Manual Pages. URL: https://spinroot.com/spin/Man/promela.html (дата обращения: 23.05.2025).

3. Документация Spin // Basic Spin Manual. URL: https://spinroot.com/spin/Man/Manual.html#S (дата обращения: 23.05.2025).

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

model.plm

```
/* Глобальная переменная с сигналом светофора для каждого направления */
mtype = { GREEN, RED };
int NS_LIGHT = RED;
int SW_LIGHT = RED;
int ED_LIGHT = RED;
int WN_LIGHT = RED;
int WD_LIGHT = RED;
int DN_LIGHT = RED;

/* Глобальное мьютексное взаимное исключение для 10 пересечений с 6
направлениями */
chan LOCK[10] = [6] of { int };

/* Глобальная переменная с наличием автомобилей для каждого направления */
bool NS_SENSE = false;
bool SW_SENSE = false;
bool ED_SENSE = false;
bool WN_SENSE = false;
bool WD_SENSE = false;
bool DN_SENSE = false;

/* Свойства безопасности */
ltl safety_NS { [] ! ((SW_LIGHT == GREEN || ED_LIGHT == GREEN || WN_LIGHT ==
GREEN || WD_LIGHT == GREEN || DN_LIGHT == GREEN) && (NS_LIGHT == GREEN)) }
ltl safety_SW { [] ! ((NS_LIGHT == GREEN || WN_LIGHT == GREEN || WD_LIGHT ==
GREEN) && (SW_LIGHT == GREEN)) }
ltl safety_ED { [] ! ((NS_LIGHT == GREEN || WN_LIGHT == GREEN || DN_LIGHT ==
GREEN) && (ED_LIGHT == GREEN)) }
ltl safety_WN { [] ! ((NS_LIGHT == GREEN || SW_LIGHT == GREEN || ED_LIGHT ==
GREEN) && (WN_LIGHT == GREEN)) }
ltl safety_WD { [] ! ((NS_LIGHT == GREEN || SW_LIGHT == GREEN || DN_LIGHT ==
GREEN) && (WD_LIGHT == GREEN)) }
ltl safety_DN { [] ! ((NS_LIGHT == GREEN || ED_LIGHT == GREEN || WD_LIGHT ==
GREEN) && (DN_LIGHT == GREEN)) }

/* Свойства живости */
ltl liveness_NS { [] ((NS_SENSE && NS_LIGHT == RED) -> <> (NS_LIGHT == GREEN)) }
ltl liveness_SW { [] ((SW_SENSE && SW_LIGHT == RED) -> <> (SW_LIGHT == GREEN)) }
ltl liveness_ED { [] ((ED_SENSE && ED_LIGHT == RED) -> <> (ED_LIGHT == GREEN)) }
```

```
ltl liveness_WN { [] ((WN_SENSE && WN_LIGHT == RED) -> <> (WN_LIGHT == GREEN)) }
ltl liveness_WD { [] ((WD_SENSE && WD_LIGHT == RED) -> <> (WD_LIGHT == GREEN)) }
ltl liveness_DN { [] ((DN_SENSE && DN_LIGHT == RED) -> <> (DN_LIGHT == GREEN)) }

/* Свойства справедливости */
ltl fairness_NS { [] <> ! ((NS_LIGHT == GREEN) && NS_SENSE) }
ltl fairness_SW { [] <> ! ((SW_LIGHT == GREEN) && SW_SENSE) }
ltl fairness_ED { [] <> ! ((ED_LIGHT == GREEN) && ED_SENSE) }
ltl fairness_WN { [] <> ! ((WN_LIGHT == GREEN) && WN_SENSE) }
ltl fairness_WD { [] <> ! ((WD_LIGHT == GREEN) && WD_SENSE) }
ltl fairness_DN { [] <> ! ((DN_LIGHT == GREEN) && DN_SENSE) }

/* Контроллер среды */
proctype env_controller() {
    do
    :: !NS_SENSE -> NS_SENSE = true;
    :: !SW_SENSE -> SW_SENSE = true;
    :: !ED_SENSE -> ED_SENSE = true;
    :: !WN_SENSE -> WN_SENSE = true;
    :: !WD_SENSE -> WD_SENSE = true;
    :: !DN_SENSE -> DN_SENSE = true;
    od
}

/* Контроллер светофора, или направления, NS */
proctype NS() {
    do
      /* В случае обнаружения автомобиля на направлении при красном сигнале
светофора ставим направление в очередь и ожидаем ее, в каждом пересечении,
      затем разрешаем проезд зеленым сигналом светофора */
    :: NS_SENSE && NS_LIGHT == RED -> {
            LOCK[0] ! 0; LOCK[0] ? <0>;
            LOCK[1] ! 0; LOCK[1] ? <0>;
            LOCK[2] ! 0; LOCK[2] ? <0>;
            LOCK[3] ! 0; LOCK[3] ? <0>;
            LOCK[4] ! 0; LOCK[4] ? <0>;
            NS_LIGHT = GREEN;
    }

      /* При включении зеленого сигнала светофора считаем проезд автомобилей
завершенным и включаем красный сигнал и убираем направление из очередей */
      :: NS_LIGHT == GREEN -> {
```

```
            NS_SENSE = false;
            NS_LIGHT = RED;
            LOCK[0] ? 0;
            LOCK[1] ? 0;
            LOCK[2] ? 0;
            LOCK[3] ? 0;
            LOCK[4] ? 0;
    }
    od
}

/* Контроллер светофора, или направления, SW */
proctype SW() {
    do
    :: SW_SENSE && SW_LIGHT == RED -> {
            LOCK[0] ! 1; LOCK[0] ? <1>;
            LOCK[5] ! 1; LOCK[5] ? <1>;
            LOCK[6] ! 1; LOCK[6] ? <1>;
            SW_LIGHT = GREEN;
    }

      :: SW_LIGHT == GREEN -> {
            SW_SENSE = false;
            SW_LIGHT = RED;
            LOCK[0] ? 1;
            LOCK[5] ? 1;
            LOCK[6] ? 1;
    }
    od
}

/* Контроллер светофора, или направления, ED */
proctype ED() {
    do
    :: ED_SENSE && ED_LIGHT == RED -> {
            LOCK[1] ! 2; LOCK[1] ? <2>;
            LOCK[7] ! 2; LOCK[7] ? <2>;
            LOCK[8] ! 2; LOCK[8] ? <2>;
            ED_LIGHT = GREEN;
    }

      :: ED_LIGHT == GREEN -> {
```

```
            ED_SENSE = false;
            ED_LIGHT = RED;
            LOCK[1] ? 2;
            LOCK[7] ? 2;
            LOCK[8] ? 2;
    }
    od
}

/* Контроллер светофора, или направления, WN */
proctype WN() {
    do
    :: WN_SENSE && WN_LIGHT == RED -> {
            LOCK[2] ! 3; LOCK[2] ? <3>;
            LOCK[5] ! 3; LOCK[5] ? <3>;
            LOCK[7] ! 3; LOCK[7] ? <3>;
            WN_LIGHT = GREEN;
    }

      :: WN_LIGHT == GREEN -> {
            WN_SENSE = false;
            WN_LIGHT = RED;
            LOCK[2] ? 3;
            LOCK[5] ? 3;
            LOCK[7] ? 3;
    }
    od
}

/* Контроллер светофора, или направления, WD */
proctype WD() {
    do
    :: WD_SENSE && WD_LIGHT == RED -> {
            LOCK[3] ! 4; LOCK[3] ? <4>;
            LOCK[6] ! 4; LOCK[6] ? <4>;
            LOCK[9] ! 4; LOCK[9] ? <4>;
            WD_LIGHT = GREEN;
    }

      :: WD_LIGHT == GREEN -> {
            WD_SENSE = false;
            WD_LIGHT = RED;
```

```
            LOCK[3] ? 4;
            LOCK[6] ? 4;
            LOCK[9] ? 4;
    }
    od
}

/* Контроллер светофора, или направления, DN */
proctype DN() {
    do
    :: DN_SENSE && DN_LIGHT == RED -> {
            LOCK[4] ! 5; LOCK[4] ? <5>;
            LOCK[8] ! 5; LOCK[8] ? <5>;
            LOCK[9] ! 5; LOCK[9] ? <5>;
            DN_LIGHT = GREEN;
    }

      :: DN_LIGHT == GREEN -> {
            DN_SENSE = false;
            DN_LIGHT = RED;
            LOCK[4] ? 5;
            LOCK[8] ? 5;
            LOCK[9] ? 5;
    }
    od
}

init {
      /* Атомарное создание всех процессов */
    atomic {
        run env_controller();
        run NS();
        run SW();
        run ED();
        run WN();
        run WD();
        run DN();
    }
}
```