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

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. Конфликтные участки пересечения проиндексированы от 0 до 5. Они представляют собой ресурс, за которые будут конкурировать процессы.

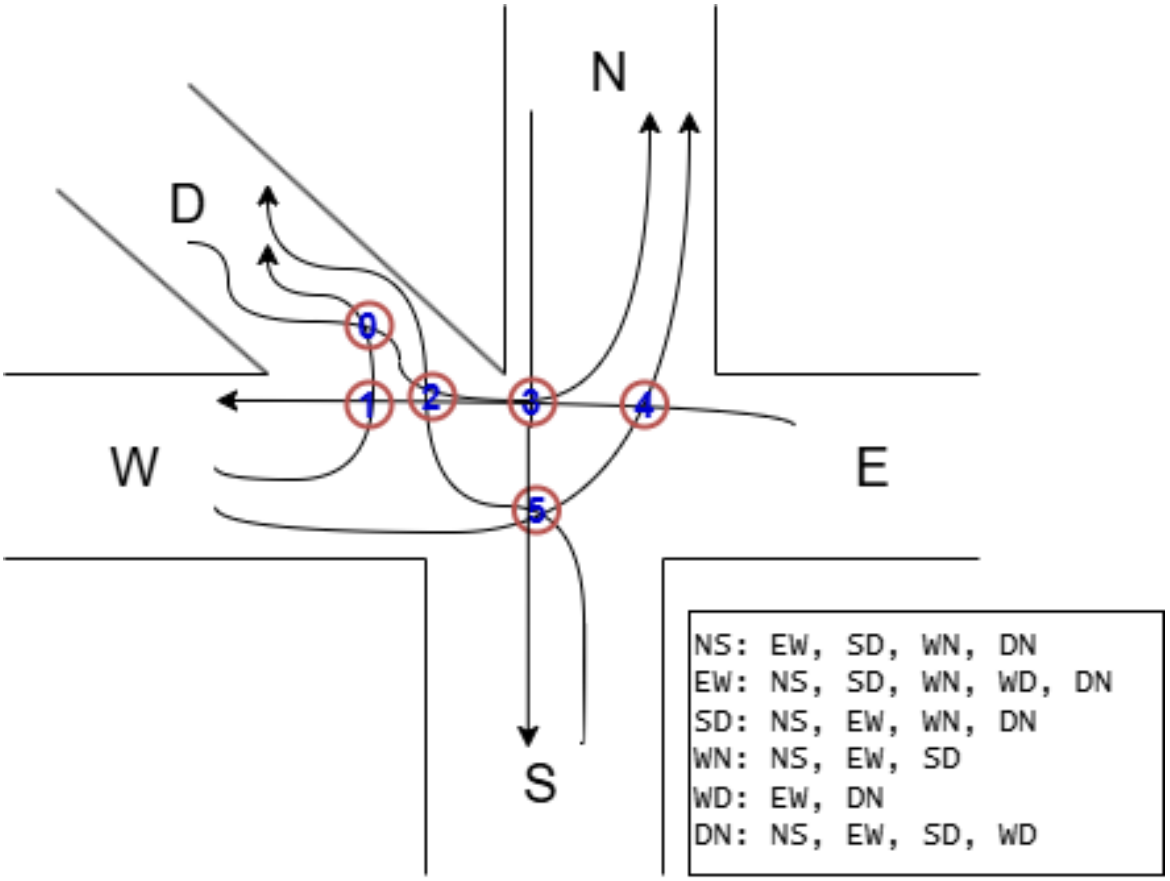

Рисунок 1 - Схема перекрёстка

Состояние модели.

Для описания состояния контроллера направления реализована структура *TLight*.

```
typedef TLight {
    int color;
    bool sense;
```

```
}
```

Она содержит цвет ***color*** светофора, флаг наличия транспорта на направлении ***sense***.

Для синхронизации процессов использован массив из N каналов ёмкостью M. Где N - количество конфликтных участков, а M - количество направлений.

```
chan lock[N] = [M] of { int };
```

Процессы.

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

Процесс трафика моделирует среду и недетерминированно активирует датчики. Если какой-то сенсор не активен - среда его активирует. Например для направления NS:

```
:: !NS.sense -> NS.sense = true
```

Каждый процесс при активации своего сенсора последовательно захватывает каждый ресурс - пересечение, необходимое для проезда по своему маршруту. Для обеспечения справедливости процесс захвата устроен следующим образом: в очередь ресурса добавляется идентификатор процесса, после чего процесс ждёт пока его идентификатор не станет первым в очереди. Таким образом захватывается каждый ресурс и происходит смена сигнала светофора. Рассмотрим фрагмент на примере направления NS:

```
:: NS.sense -> {
    lock[3] ! NSid;
    lock[3] ? <NSid>;
    lock[5] ! NSid;
    lock[5] ? <NSid>;
```

Далее моделируется отключение сенсора, смена сигнала и некоторый проезд по направлению. После включения зелёного сигнала, можно сделать допущение, что проходит некоторое время для проезда автомобилей, перед тем как он выключится. После включения красного сигнала, занятые ресурсы освобождаются. Рассмотрим на примере направления NS:

```
NS.sense = false;
NS.color = GREEN;
/* Происходит проезд */
NS.color = RED;
lock[3] ? NSid;
lock[5] ? NSid;
```

Для защиты от блокировок (deadlock) все процессы захватывают ресурсы в одном и том же порядке - от меньшего индекса пересечения к большему, и освобождают их в таком же порядке.

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=4096 -DNFAIR=3 -O2 -w*.

Для запуска модели использовались: *-m1000000000 -a -f -n -c1*.

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

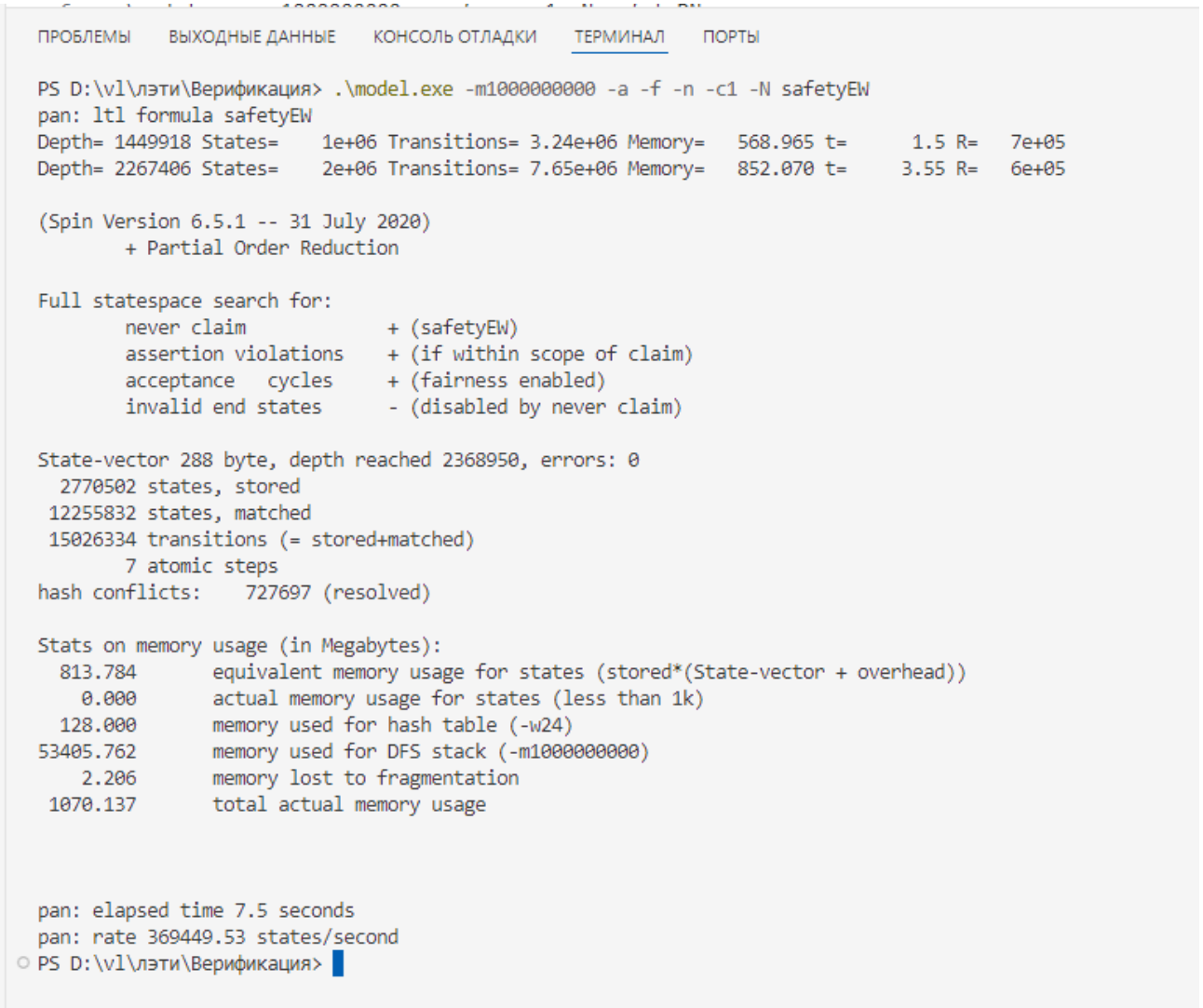

Рисунок 2 - Верификация условия безопасности для направления EW

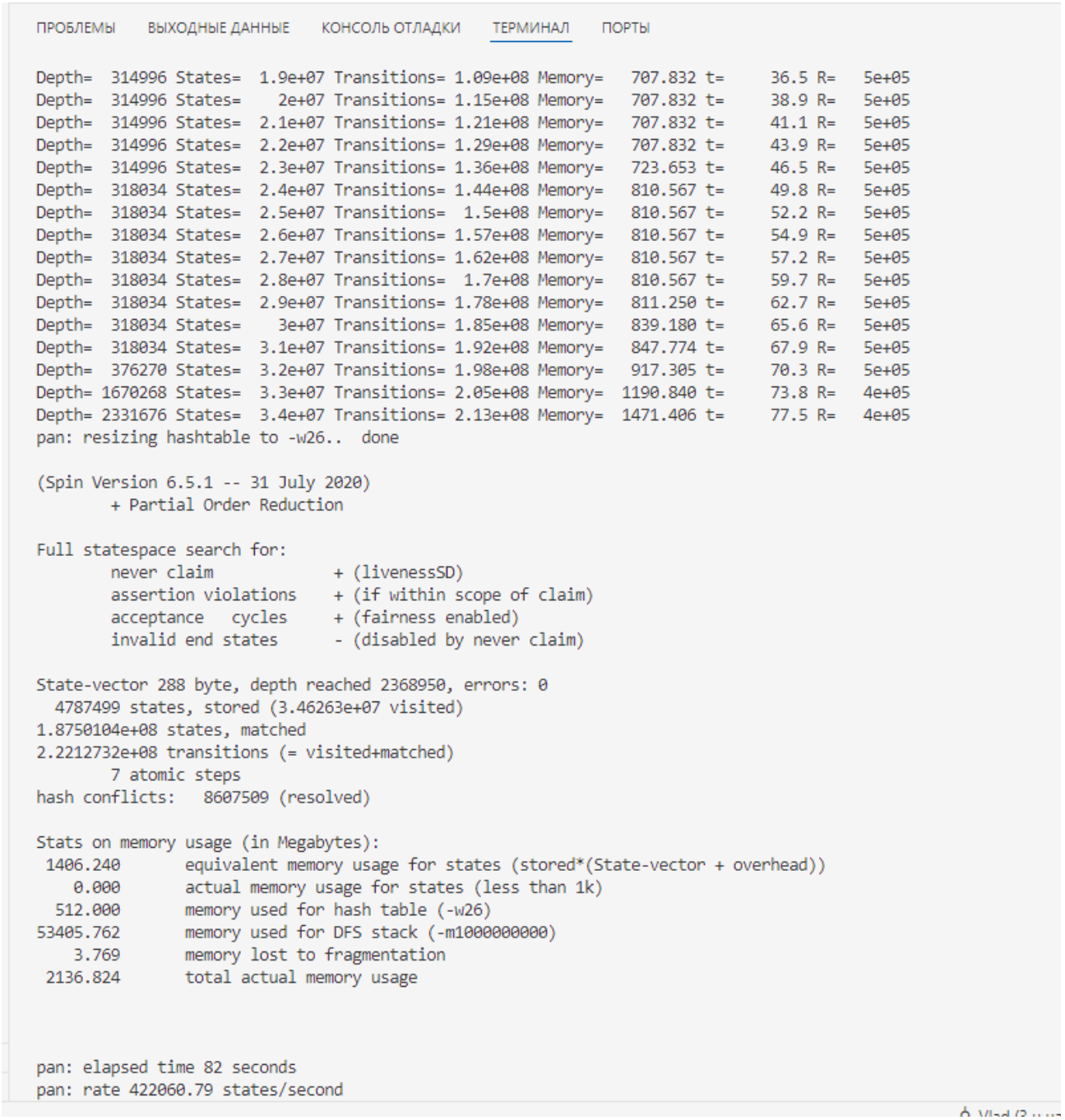

Рисунок 3 - Верификация условия живости для направления SD

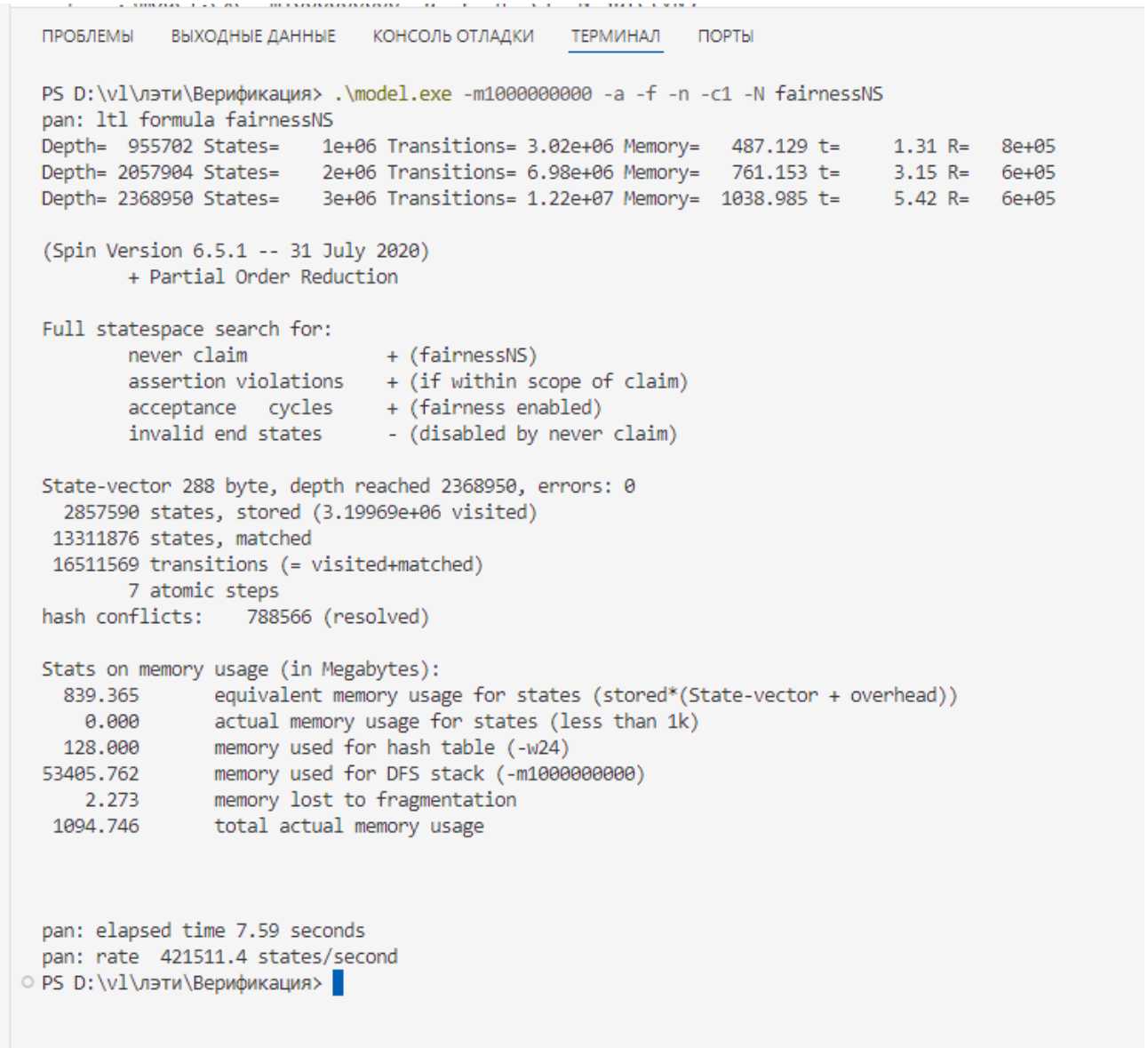

Рисунок 4 - Верификация условия справедливости для направления NS

Выводы.

В результате выполнения работы была спроектирована модель системы сложного перекрёстка. Реализована программа на языке promela, описывающая данную модель. С помощью параллельных процессов смоделирован траффик и контроллеры светофоров заданных направлений. Для защиты от блокировок использовался подход со взятием ресурсов в одинаковой последовательности. Для защиты от голодания процессов, использована очередь на основе канала.

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;
}

mtype = { GREEN, RED }

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

// Количество пересечений
#define N 6
// Количество направлений
#define M 6
// Id направлений
#define NSid 0
#define EWid 1
#define SDid 2
#define WNid 3
#define WDid 4
#define DNid 5

/* Очереди мьютексов для пересечений */
chan lock[N] = [M] of { int };

/* 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) }

init {
    atomic {
        NS.color = RED;
        EW.color = RED;
        SD.color = RED;
        WN.color = RED;
        WD.color = RED;
        DN.color = RED;
        NS.sense = false;
        EW.sense = false;
        SD.sense = false;
        WN.sense = false;
        WD.sense = false;
        DN.sense = false;

        run Traffic();
        run NSproc();
        run EWproc();
        run SDproc();
        run WNproc();
        run WDproc();
        run DNproc();
    }
}

/* Моделирование траффика. Если не активирован сенсор - активируем
*/
proctype Traffic() {
    do
    :: !NS.sense -> NS.sense = true
    :: !EW.sense -> EW.sense = true
    :: !SD.sense -> SD.sense = true
    :: !WN.sense -> WN.sense = true
    :: !WD.sense -> WD.sense = true
    :: !DN.sense -> DN.sense = true
    od
}

/* Процесс для каждого направления (сфетофора) */
proctype NSproc() {
```

```
    do
    :: NS.sense -> {
        // Добавляем наш id в канал
        lock[3] ! NSid;
        // Ждём пока до него дойдёт очередь
        lock[3] ? <NSid>;
        // Аналогично
        lock[5] ! NSid;
        lock[5] ? <NSid>;

        // Выключаем сенсор и устанавливаем зелёный свет
        NS.sense = false;
        NS.color = GREEN;

        /* Происходит проезд */

        /* Устанавливаем красный свет */
        NS.color = RED;
        // Освобождаем канал от своего id
        lock[3] ? NSid;
        lock[5] ? NSid;
    }
    od
}

proctype EWproc() {
    do
    :: EW.sense -> {
        lock[1] ! EWid;
        lock[1] ? <EWid>;
        lock[2] ! EWid;
        lock[2] ? <EWid>;
        lock[3] ! EWid;
        lock[3] ? <EWid>;
        lock[4] ! EWid;
        lock[4] ? <EWid>;

        EW.sense = false;
        EW.color = GREEN;
        /* */
        EW.color = RED;
        lock[1] ? EWid;
        lock[2] ? EWid;
        lock[3] ? EWid;
        lock[4] ? EWid;
    }
    od
}

proctype SDproc() {
    do
    :: SD.sense -> {
        lock[2] ! SDid;
```

```
            lock[2] ? <SDid>;
            lock[5] ! SDid;
            lock[5] ? <SDid>;

            SD.sense = false;
            SD.color = GREEN;
            /* */
            SD.color = RED;
            lock[2] ? SDid;
            lock[5] ? SDid;
        }
        od
}

proctype WNproc() {
        do
        :: WN.sense -> {
            lock[4] ! WNid;
            lock[4] ? <WNid>;
            lock[5] ! WNid;
            lock[5] ? <WNid>;

            WN.sense = false;
            WN.color = GREEN;
            /* */
            WN.color = RED;
            lock[4] ? WNid;
            lock[5] ? WNid;
        }
        od
}

proctype WDproc() {
        do
        :: WD.sense -> {
            lock[0] ! WDid;
            lock[0] ? <WDid>;
            lock[1] ! WDid;
            lock[1] ? <WDid>;

            WD.sense = false;
            WD.color = GREEN;
            /* */
            WD.color = RED;
            lock[0] ? WDid;
            lock[1] ? WDid;
        }
        od
}

proctype DNproc() {
        do
        :: DN.sense -> {
```

```
            lock[0] ! DNid;
            lock[0] ? <DNid>;
            lock[2] ! DNid;
            lock[2] ? <DNid>;
            lock[3] ! DNid;
            lock[3] ? <DNid>;

            DN.sense = false;
            DN.color = GREEN;
            /* */
            DN.color = RED;
            lock[0] ? DNid;
            lock[2] ? DNid;
            lock[3] ? DNid;
        }
        od
}
```