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.

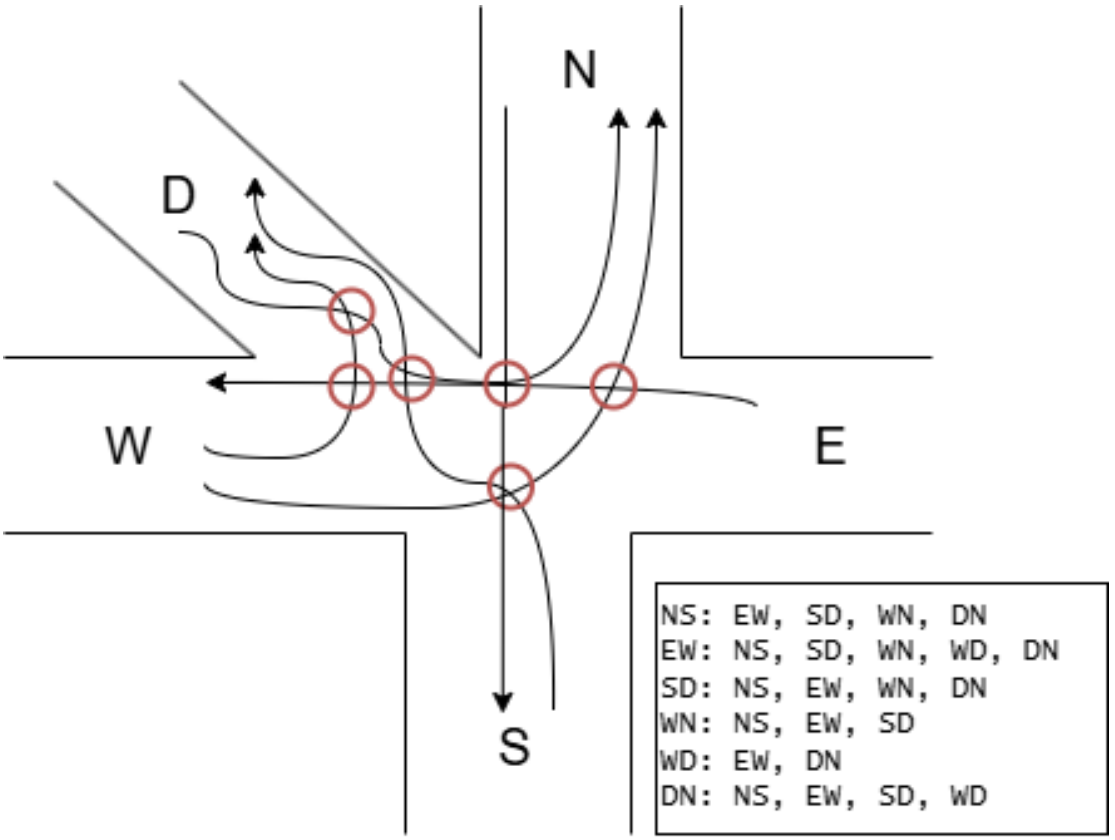

Рисунок 1 - Схема перекрёстка

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

Следующем 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.

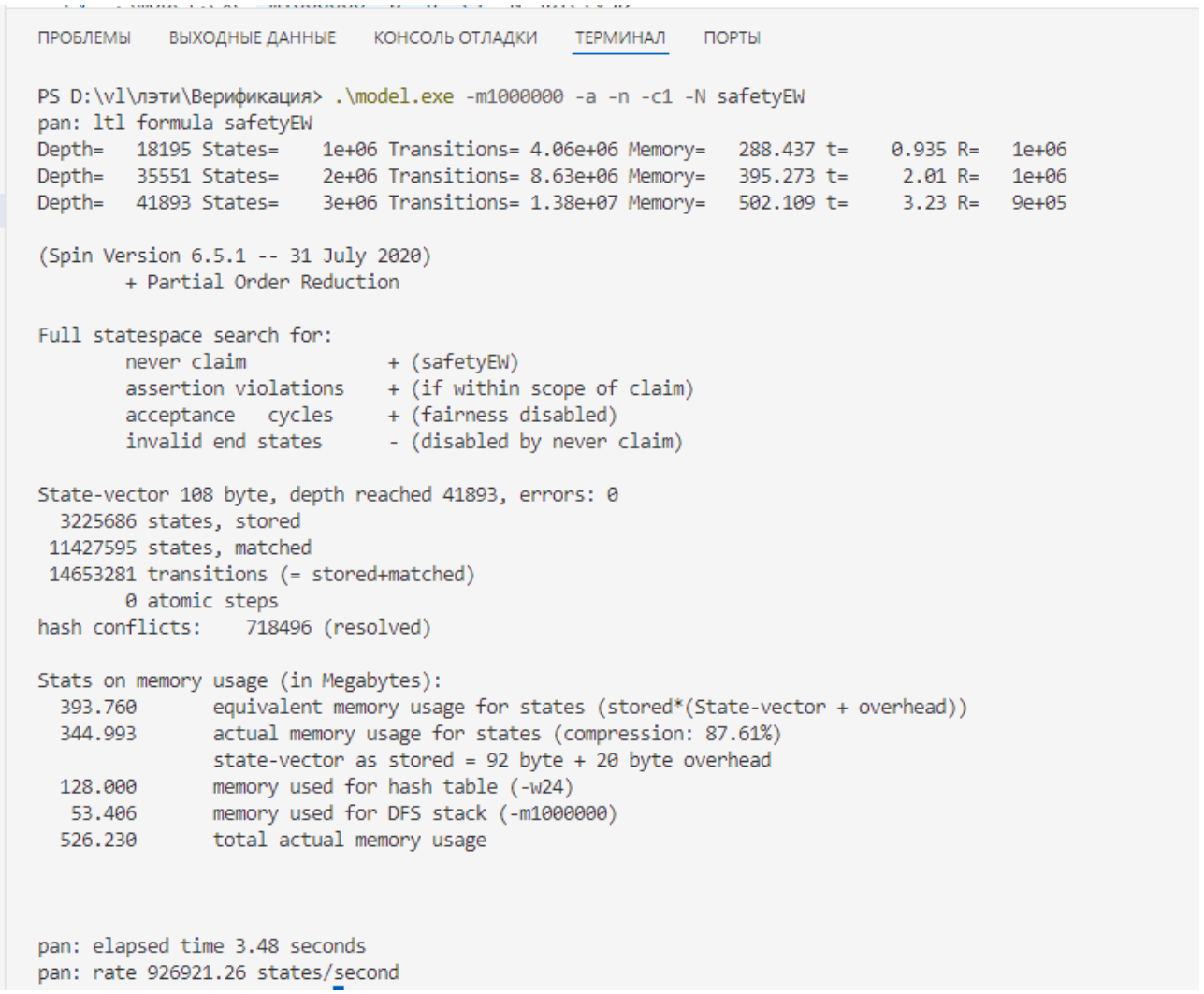

Рисунок 2 - Верификация условия безопасности для направления EW

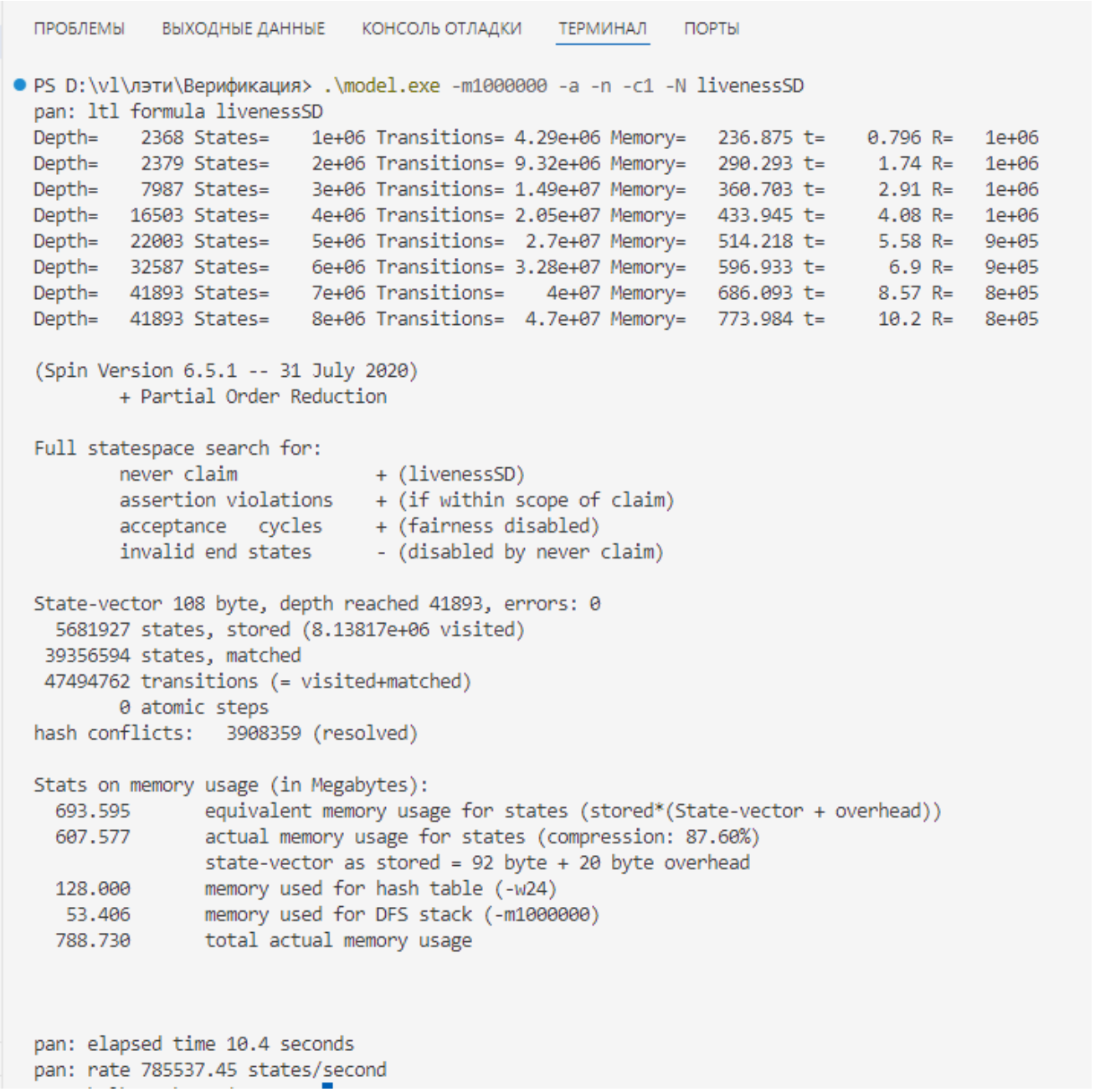

Рисунок 3 - Верификация условия живости для направления SD

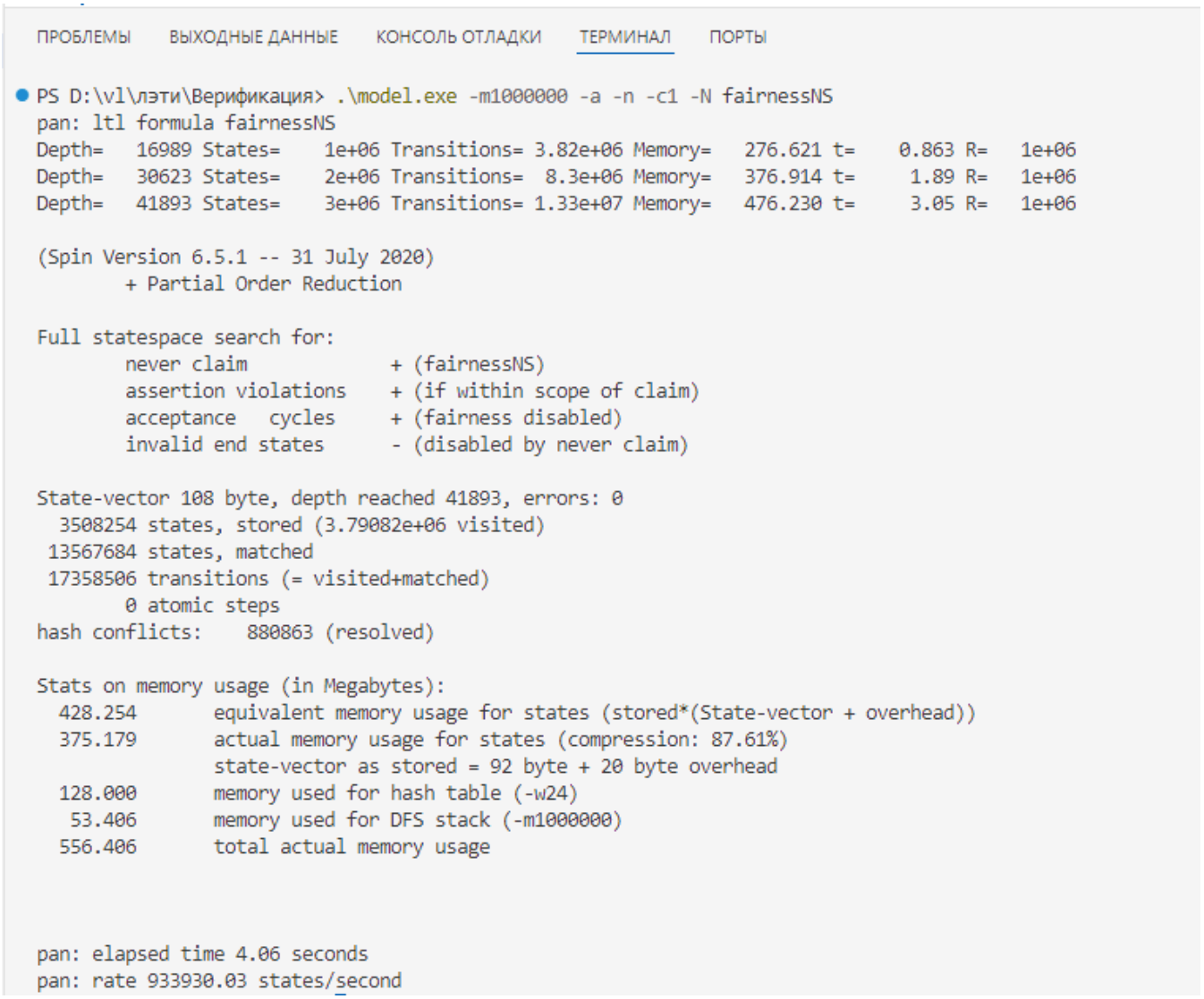

Рисунок 4 - Верификация условия справедливости для направления NS

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.

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 траффика */
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
```

```
    }

    /* 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 (сфетофора) */
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
}
```