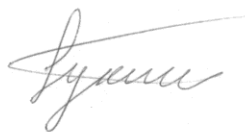


Минобрнауки России
Санкт-Петербургский государственный политехнический университет
Институт информационных технологий и управления
Кафедра «Информационные и управляющие системы»

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.....	4
Реализация.....	5
Основная идея.....	5
Основные обозначения	7
Реализация процессов	8
LTL свойства.....	10
Безопасность	10
Живость и справедливость	11
Описание LTL свойств на в системе Spin.....	12
Результаты моделирования	13
Выводы	16
Список литературы	18
Приложения	19

Список изображений:

Рисунок 1. Графическое изображения варианта задания	4
---	---

Введение

От программных систем все в большей степени каждодневно зависят жизнь и здоровье людей, однако программирование до сих пор остается единственной областью инженерной деятельности, где разработчик фактически не может гарантировать качество своей работы^[1].

Верификация (model checking) – это набор формальных приемов и методов подтверждения того, что разрабатываемая система удовлетворяет формальным установленным требованиям (формальной спецификации)^[2].

В данной курсовой работе рассматривается модель контроллера управления движением на дорожном перекрестке. Необходимо описать модель и проверить ее корректность при выполнении следующих условий:

- Каждое направление контролирует отдельный светофор
- Поведение светофоров описывается параллельными процессами
- Необходимо моделировать появление машин
- Алгоритм управления движением не должен определяться заранее заданным порядком переключения светофоров
- В системе для каждого из направлений присутствуют датчики, фиксирующие наличие автомобилей.

Для model checking используется система Spin. Данная система позволяет описывать модель на языке Promela и задавать требуемые свойства системы при помощи выражений LTL.

Постановка задачи

Вариант (1, 12, 15).

Пересечения: $\{(NS, WN), (NE, EW), (SW, ES)\}$ (рисунок. 1)

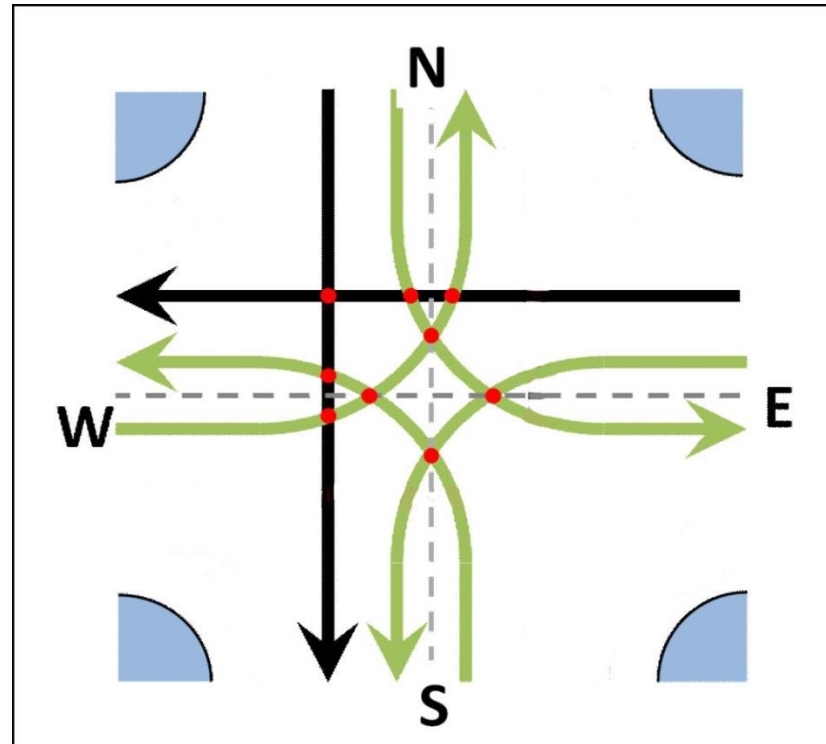


Рисунок 1. Графическое изображение варианта задания

Для заданного варианта необходимо описать набор заданных в методическом пособии свойств LTL на языке Promela (см. пункт LTL свойства), которые определяют корректность работы модели, а так же описать саму модель на языке Promela.

Реализация

Основная идея

Исследуемая модель можно рассматривать как задачу синхронизации доступа к разделяемым ресурсам. При этом разделяемыми ресурсами выступают конкурентные пересечения направлений движения (под конкуренцией в данном случае понимается невозможность одновременного движения по двум направлениям, образующим пересечение). Для реализации данной модели были выделены следующие положения:

- 1) Каждый контроллер светофора описывается отдельным процессом
- 2) Сигнал светофора может быть двух видов: красный (запрещающий движение) – зеленый (разрешающий движение)
- 3) Датчик движения определяет наличие машины перед светофором
- 4) Появление машин (трафик) генерируется внешним, по отношению к контроллерам, процессом
- 5) Появление машин происходит по всем направлениям независимо друг от друга (нет заранее определенной очередности возникновения машин на направлениях)

Разделяемые ресурсы (конкурентные пересечения) описываются каналами единичной емкости и принимающие элементы типа **bool**. При этом захват ресурсы означает получение из канала значения **true**. Для упрощения реализации некоторые каналы задают не одно пересечение, а несколько. Объединение пересечений при этом происходит по принципу взаимоисключения, то есть объединяются пересечения тех направлений движения, которые являются заведомо взаимоисключающими.

Описание каналов на языке Promela:

```
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};
```

В объявлении используются следующие обозначения:

- **chan** – ключевое слово языка Promela, означает что данная переменная является каналом
- Название переменной (например **NS_WN_EW**) – формируется из названий направлений (аналогично рисунку 1), образующих пересечения
- **[1]** – глубина (емкость) канала - определяет количество сообщений, которые могут одновременно находиться в канале.
- **Of {bool}** – определяет тип данных, передаваемых каналом. В данном случае это булливый тип

Для обеспечения безопасного проезда, перед переключением сигнала светофора на зеленый (разрешающий) сигнал должен произойти захват всех каналов, описывающих конкурентные пересечения для данного направления.

Описания процессов:

- **Init** – процесс инициализации. В нем каналы заполняются инициализирующими значениями, а так же запускаются процессы контроллеров и генератора трафика
- **proctype gen_t ()** – процесс генерации трафика
- процессы контроллеров именуются по аббревиатуре направления (например, **proctype NS ()**)

Основные обозначения

В данном разделе описываются обозначения, принятые в программе:

- 1) Типы сигналов светофоров (**mtype** = {**Red**, **Green**}) – принимают значения **Red** и **Green**, что является запрещающим и разрешающим движение сигналами соответственно
- 2) Сигналы светофоров описываются следующими переменными:

NS_L, WN_L, NE_L, EW_L, ES_L, SW_L

Где символ L обозначает light – свет светофора, а первые две буквы определяют направление, которое контролирует данный светофор (аббревиатура направления аналогично описанной в задании – рисунок 1)

- 3) Датчики наличия машин перед светофором описываются следующими переменными:

NS_S, WN_S, NE_S, EW_S, ES_S, SW_S

Где символ S обозначает sensor – датчик наличия машины (аббревиатура направления аналогично описанной в предыдущем пункте)

- 4) Метки, используемые для верификации:

****True** – задает точку выполнения программы, в которой датчик движения по направлению ****** определяет наличие машины

****False** – задает точку выполнения программы, в которой датчик движения по направлению ****** не определяет наличия машины

green = задает точку выполнения программы, в которой светофор того контролера, в теле процесса которого сигнал светофора зеленый

red – аналогично предыдущему пункту, только сигнал светофора – красный

****** - аббревиатура направления (например, NS)

Реализация процессов

В данном разделе будут описаны идеи реализации процессов контролеров и генератора трафика:

1) процесс генерации трафика (gen_t)

Данный процесс представляет собой бесконечный цикл (do), для которого определены следующие условия (рассматриваются на примере одного направления - NS, остальные аналогично):

:: (!NS_S) ->

NSTrue: NS_S = true;

:: (NS_L==Green && NS_S) ->

NSFalse: NS_S = false;

Логика данного условия такова:

Если датчик светофора показывает отсутствие машин, то датчик будет переключен (что соответствует появлению машины и срабатыванию датчика). Второе условие – если сигнал светофора зеленый (что соответствует ситуации, когда машины проезжают по данному направлению), то датчик будет переключен в состояние, соответствующее окончанию потока машин.

2) процессы контролера светофора

Данные процессы аналогичны для всех направлений, поэтому мы рассмотрим один процесс на примере направления NS. Указанные процессы представляют собой бесконечный цикл, в котором при соблюдении условия сработавшего датчика движения (фиксируется наличие ожидающей на данном направлении машины) контролер захватывает ресурсы рекуррентных направлений (ожидая их освобождения), после чего открывает проезд (сигнал светофора переключается на зеленый). Далее контроллер ожидает окончания потока машин (датчик движения перестает фиксировать наличие машин), переключает сигнал светофора в красный и освобождает ресурсы.


```
proctype NS ()
{
  end:
  do
    :: NS_S ->
      NS_WN_EW ? true; NS_WN_SW ? true;
      NS_L = Green;
  green:
    if
      :: !NS_S -> skip;
    fi;
    NS_L = Red;
  red:
    NS_WN_EW ! true; NS_WN_SW ! true;
  od;
}
```

LTL свойства

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 (safety)** для каждого направления 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». Далее представлены свойства безопасности для каждого направления, сформулированные в терминах системы Spin. Обозначения соответствует описанным ранее.

- 1) **NS** - $\text{ltl } p0_1 \{ \square !pNS_S \}$
 $pNS_S (NS_L == Green \ \&\& \ WN_L == Green \ \&\& \ SW_L == Green \ \&\& \ EW_L == Green)$
- 2) **WN** – $\text{ltl } p0_2 \{ \square !pWN_S \}$
 $pWN_S (WN_L == Green \ \&\& \ NS_L == Green \ \&\& \ NE_L == Green \ \&\& \ SW_L == Green \ \&\& \ EW_L == Green)$
- 3) **NE** - $\text{ltl } p0_3 \{ \square !pEW_S \}$
 $pEW_S (EW_L == Green \ \&\& \ NE_L == Green \ \&\& \ WN_L == Green \ \&\& \ NS_L == Green)$
- 4) **EW** - $\text{ltl } p0_4 \{ \square !pNE_S \}$
 $pNE_S (NE_L == Green \ \&\& \ WN_L == Green \ \&\& \ ES_L == Green \ \&\& \ EW_L == Green)$

5) **SW** - ltl p0_5 {[[] !pSW_S}

pSW_S (SW_L==Green && ES_L==Green && WN_L==Green &&
NS_L==Green)

6) **ES** - ltl p0_6 {[[] !pES_S}

pES_S (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 данного направления горит красным и датчик показывает наличие машин, то когда-то в будущем данный светофор будет гореть зеленым». А правила **справедливости (fairness)** - «Невозможна такая ситуация, что в данном направлении движется непрерывный поток машин(т.е. светофор должен неопределенно часто менять свой сигнал на запрещающий)». «Данное направление» подразумевает собой любое из рассмотренных выше.

Доказывать свойства **fairness** и **liveness** отдельно для данной модели не представляется возможным из-за того, что построенная модель не является справедливой **по построению**. Для верификации такой модели можно применяться следующие подходы^{[5][6][7][8]}:

- Перестроить модель таким образом, чтобы она стала справедливой
- Исключить трассы, не удовлетворяющие требованиям справедливости из рассмотрения

В данном случае проще применить второй подход (исключение трасс), так как он не требует перестроения модели. 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».

Описание LTL свойств на в системе Spin

Для описания свойств корректности системы использовались следующие переменные (рассматривается вариант для направления NS, остальные направления описываются аналогично):

```
#define pNS_S (NS@green && WN@green && SW@green && EW@green)
#define pNS_L (gen_t@NSTrue && (NS@red))
#define qNS_L (NS@green)
#define pNS_F ((NS@green) && gen_t@NSTrue)
```

pNS_S – описывает логическое выражение свойства **безопасности** для направления **NS**

pNS_L – описывает логическое выражение левой части в свойстве **живучести** для направления **NS**

qNS_L – описывает логическое выражение следствия в свойстве **живучести** для направления **NS**

pNS_F – описывает логическое выражение свойства **справедливости** для направления **NS**

С учетом введенных обозначений и рассмотренных ранее обозначений меток ltl-формулы для направления NS будут выглядеть следующим образом (остальные направления описываются аналогично):

- 1) Безопасность

ltl pS_NS {[] !pNS_S}

- 2) Живучесть при условии справедливости

ltl pF_NS {[]<> (([]<> !(pNS_F)) -> ([] (pNS_L -> <> qNS_L)))}

Результаты моделирования

Моделирование показало, что построенная модель удовлетворяет всем заявленным для нее свойствам (безопасность, живучесть и справедливость). Ниже представлен вывод системы Spin для одного из свойств (безопасность для направления NS – pNS_S) ^{[3][4]}:

```
gcc -DMEMLIM=3500 -O2 -DXUSAFE -o pan pan.c  
./pan -m10000000 -a -N pS_NS
```

Ключи компиляции:

- DMEMLIM – количество отведенной под верификацию памяти в мегабайтах
- O2 – режим оптимизации программы
- DXUSAFE – исключает проверку channel assertions – свойство доступа процессов к каналам
- a – включает проверку отсутствия циклов бесконечного повторения операторов с меткой assert
- o – определяет имя объектного файла (создается после компиляции)
- m – определяет глубину верификации (число шагов)
- N – определяет используемое в данном случае ltl правило

Pid: 3262 - Идентификатор процесса

**Depth=1106689 States=1e+06 Transitions=3.2e+06 Memory=746.140 t=1.51
R=7e+05**

**Depth=1861295 States=2e+06 Transitions=6.91e+06 Memory=830.124 t=3.27
R=6e+05**

**Depth=2454143 States=3e+06 Transitions=1.08e+07 Memory=914.109 t=5.14
R=6e+05**

**Depth=2878837 States=4e+06 Transitions=1.48e+07 Memory=998.093 t=7.09
R=6e+05**

**Depth=3364751 States=5e+06 Transitions=1.87e+07 Memory=1082.077 t=8.98
R=6e+05**

**Depth=3731131 States=6e+06 Transitions=2.27e+07 Memory=1166.062 t=11
R=5e+05**

**Depth=3764329 States=7e+06 Transitions=2.87e+07 Memory=1249.948 t=14
R=5e+05**

Данный вывод показывает глубину верификации, количество состояний, количество переходов, используемую память, время и редукцию для каждого шага

unreached in init

(0 of 15 states)

unreached in proctype gen_t

var1.pml:88, state 28, "-end-"

(1 of 28 states)

unreached in proctype NS

var1.pml:107, state 15, "-end-"

(1 of 15 states)

unreached in proctype WN

var1.pml:127, state 17, "-end-"

(1 of 17 states)

unreached in proctype NE

var1.pml:147, state 15, "-end-"

(1 of 15 states)

unreached in proctype EW

var1.pml:167, state 15, "-end-"

(1 of 15 states)

unreached in proctype ES

var1.pml:187, state 15, "-end-"

(1 of 15 states)

unreached in proctype SW

var1.pml:207, state 15, "-end-"

(1 of 15 states)

unreached in claim pS_NS

_spin_nvr.tmp:8, state 10, "-end-"

(1 of 10 states)

Данная информация показывает недостижимые состояния для каждого из процессов. В данном случае, так как процессы контролеров и генерации трафика бесконечны, то конечное состояние недостижимо

pan: elapsed time 17.3 seconds

No errors found -- did you verify all claims?

В последнем выводе можно видеть затраченное на проверку данного свойства время и вывод, отвечающий отсутствию ошибок и контрпримера.

Выводы

В рамках данной работы была построена модель контролера светофоров, а так же была проведена верификация данной модели на основе определенных в задании свойств. Проведенная верификация показала, что построенная модель удовлетворяет все требуемым свойствам. В процессе были получены следующие знания/опыт:

- Построение распределенных моделей, родственных задаче синхронизации ресурсов, на языке Promela
- Формулирование свойств построенной модели на языке темпоральной логики
- Перевод свойств модели, описанных языком темпоральной логики в термины системы Spin
- Возможности системы Spin по верификации распределенных систем
- Возможности языка Promela по описанию распределенных моделей

Последний пункт рассмотрим более подробно. В чем же отличия языка Promela от традиционных объектно-ориентированных или функциональных языков? Язык Promela обладает более узким по сравнению с традиционными языками набором примитивов: нет возможности создавать процедур/функции, нет обработки исключений (exceptions), меньший набор типов данных, циклы представлены единственной конструкцией и.т.д. Однако стоит заметить, что возможности языка Promela и его правила позволяют с большим удобством описывать распределенные системы, базирующиеся на одновременно выполняющихся процессах. Примитивом выполнения в данном языке является процесс, что уже само по себе накладывает отпечаток распределенности на программу.

Система Spin, используемая в данной работе, представляет широкие возможности для верификации моделей описанных на языке Promela: есть возможность вручную задавать параметры компиляции, ограничения по памяти и глубине верификации. Все найденные ошибки могут быть

воспроизведены. Весь вывод верификации может быть легко сохранен в файл. Так же стоит отметить возможность построения схемы-автомата системы, что является наглядным изображением модели.

Можно говорить об успешном завершении данной курсовой работы

Список литературы

- 1) Карпов Ю.Г. Новая жизнь верификации / Издательство Открытые системы, 2012-03
- 2) Карпов Ю.Г., Шошмина И.В. /Верификация распределенных систем – СПб.: Издательство Политехнического университета, 2011.
- 3) Holzmann G.J. The Model Checker Spin // IEEE Trans. on Software Engineering, Vol. 23, No. 5, май 1997, с. 279-295.
- 4) Holzmann G.J. The SPIN Model Checker: Primer and Reference Manual – Addison Wesley, 2004. – 610 с.
- 5) Thomas Wahl – Fairness and Liveness URL:
<http://www.ccs.neu.edu/home/wahl/Publications/fairness.pdf>
- 6) Concurrent programming lab2 URL:
http://www2.compute.dtu.dk/courses/02158/sol_cplab2.html
- 7) 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>
- 8) AG-Wehrheim – Verification with SPIN URL:
http://www.cs.uni-paderborn.de/fileadmin/Informatik/AG-Wehrheim/Lehre/SS09/Model_Checking/Slides/11May09.pdf

Приложения

В приложения к отчету находится полный код рассматриваемой модели

```
/* 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;
```

```
/* 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 ()
{
    end: do
        :: (!NS_S) ->
        NSTrue:  NS_S = true;
        :: (NS_L==Green) ->
        NSFalse: NS_S = false;
        :: (!WN_S) ->
        WNTrue:  WN_S = true;
        :: (WN_L==Green) ->
        WNFalse: WN_S = false;
        :: (!NE_S) ->
        NETTrue: NE_S = true;
        :: (NE_L==Green) ->
        NEFalse: NE_S = false;
        :: (!EW_S) ->
        EWTrue:  EW_S = true;
        :: (EW_L==Green) ->
        EWFalse: EW_S = false;
        :: (!ES_S) ->
        ESTTrue: ES_S = true;
        :: (ES_L==Green) ->
        ESFalse: ES_S = false;
        :: (!SW_S) ->
        SWTrue:  SW_S = true;
        :: (SW_L==Green) ->
        SWFalse: SW_S = false;
    od;
}

```

```

/* NS controller */
proctype NS ()
{
    end:
        do
            :: NS_S ->
                /* Wait for resources */
                NS_WN_EW ? true; NS_WN_SW ? true;
                NS_L = Green;
        green:
            if
                /* Wait for end of car queue */
                :: !NS_S -> skip;
            fi;
            NS_L = Red;
        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;
        green:

```

```

    if
        /* Wait for end of car queue */
        :: !WN_S -> skip;
    fi;
    WN_L = Red;
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;
green:
        if
            /* Wait for end of car queue */
            :: !NE_S -> skip;
        fi;
        NE_L = Red;
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;
        green:
            if
                /* Wait for end of car queue */
                :: !EW_S -> skip;
            fi;
            EW_L = Red;
        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;
        green:

```



```

    if
        /* Wait for end of car queue */
        :: !ES_S -> skip;
    fi;
    ES_L = Red;
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;
    green:
        if
            /* Wait for end of car queue */
            :: !SW_S -> skip;
        fi;
        SW_L = Red;
    red:
        NS_WN_SW ! true; ES_SW ! true;
        od;
}

```

```

#define pNS_S (NS@green && WN@green && SW@green && EW@green)
#define pWN_S (WN@green && NS@green && NE@green && SW@green &&
EW@green)
#define pEW_S (EW@green && NE@green && WN@green && NS@green)
#define pNE_S (NE@green && WN@green && ES@green && EW@green)
#define pSW_S (SW@green && ES@green && WN@green && NS@green)
#define pES_S (SW@green && ES@green && WN@green && NS@green)

#define pNS_L (gen_t@NSTrue && (NS@red))
#define qNS_L (NS@green)

#define pWN_L (gen_t@WNTrue && (WN@red))
#define qWN_L (WN@green)

#define pEW_L (gen_t@EWTrue && (EW@red))
#define qEW_L (EW@green)

#define pNE_L (gen_t@NETrue && (NE@red))
#define qNE_L (NE@green)

#define pSW_L (gen_t@SWTrue && (SW@red))
#define qSW_L (SW@green)

#define pES_L (gen_t@ESTrue && (ES@red))
#define qES_L (ES@green)

#define pNS_F ((NS@green) && gen_t@NSTrue)
#define pWN_F ((WN@green) && gen_t@WNTrue)
#define pEW_F ((EW@green) && gen_t@EWTrue)
#define pNE_F ((NE@green) && gen_t@NETrue)

```

```
#define pSW_F ((SW@green) && gen_t@SWTrue)
```

```
#define pES_F ((ES@green) && gen_t@ESTrue)
```

```
/*Safety*/
```

```
ltl pS_NS {[ [!pNS_S]
```

```
ltl pS_WN {[ [!pWN_S]
```

```
ltl pS_EW {[ [!pEW_S]
```

```
ltl pS_NE {[ [!pNE_S]
```

```
ltl pS_ES {[ [!pES_S]
```

```
ltl pS_SW {[ [!pSW_S]
```

```
/*Fairness -> Liveness*/
```

```
ltl pF_NS {[<> (([<> !(pNS_F)) -> ([ (pNS_L -> <> qNS_L)))]}
```

```
ltl pF_WN {[<> (([<> !(pWN_F)) -> ([ (pWN_L -> <> qWN_L)))]}
```

```
ltl pF_EW {[<> (([<> !(pEW_F)) -> ([ (pEW_L -> <> qEW_L)))]}
```

```
ltl pF_NE {[<> (([<> !(pNE_F)) -> ([ (pNE_L -> <> qNE_L)))]}
```

```
ltl pF_ES {[<> (([<> !(pES_F)) -> ([ (pES_L -> <> qES_L)))]}
```

```
ltl pF_SW {[<> (([<> !(pSW_F)) -> ([ (pSW_L -> <> qSW_L)))]}
```