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

ОТЧЕТ

по индивидуальному домашнему заданию
по дисциплине «Верификация распределенных алгоритмов»
Тема: Разработка контроллера светофоров и его верификация
Вариант: 4

Студент гр. 9303 _____ Игнашов В.М.

Преподаватель _____ Шошмина И.В.

Санкт-Петербург
2024

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Целью работы является разработка модели контроллера перекрестка, регулируемого светофором, на языке Promela. Модель должна обрабатывать потоки машин по нескольким пересекающимся направлениям (в случае если направления движений не пересекаются – допустимо одновременное выполнение потоков). Предполагается, что потоки машин недетерминированные, процессы модели обрабатывают их параллельно для исключения последовательной обработки потоков. Необходимо реализовать модель таким образом, чтобы она соответствовала проверяемым требованиям: безопасность движения, живость движения и справедливость движения.

1. ПОСТРОЕНИЕ МОДЕЛИ

1.1. Описание задачи

Вариант: 4.

Четыре пересечения: NS/ED, SD/WN, SD/DN, WN/DE

Схема сложного перекрестка с указанными выше пересечениями представлена на рисунке 1. Синими точками отмечены конфликтующие направления из условия, красными – конфликтующие направления вне условия задачи.

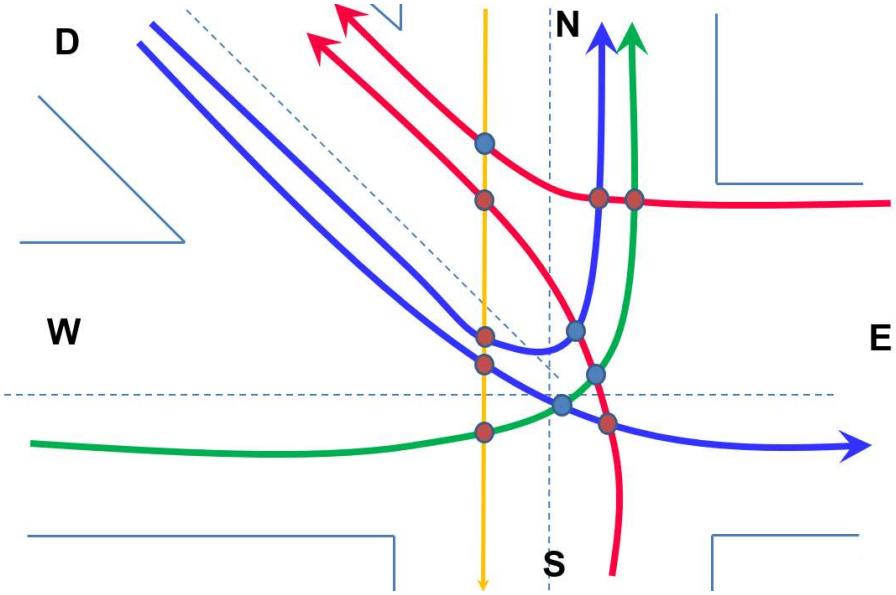

Рисунок 1. Схема перекрестка

Информация о том, какие направления должны быть закрыты при потоке машин для каждого направления, представлена в таблице 1.

Таблица 1. Конфликтующие направления

	NS	WN	SD	ED	DE	DN
NS		+	+	+	+	+
WN	+		+	+	+	
SD	+	+			+	+
ED	+	+				+
DE	+	+	+			
DN	+		+	+		

Дополнительные условия, которые необходимо учесть:

- Машины на каждом направлении движутся независимо
- Появление машины в каждом направлении регистрируется своим независимым датчиком движения

Исходный код реализованной модели представлен в приложении А.

1.2. Описание состояний

Состояние каждого контроллера направления описывается реализованной структурой Direction, имеющей три булевых поля

```
typedef Direction {
	bool cars_in_direction = 0;
	bool open = 0;
	bool should_be_open_next = 0;
};
```

- cars_in_direction – маркер, обозначающий наличие машин на этом направлении. Параметр использует логический тип данных в связи с отсутствием необходимости контролировать количество машин в потоке – независимо от их количества они все выполнят движение в случае открытого направления;
- open – маркер, обозначающий сигнал светофора (0 при закрытой дороге, 1 при открытой);
- should_be_open_next – маркер, обозначающий готовность к открытию движения. Обновляется значение на 1 в случае, когда на каждом направлении действия были выполнены. Таким образом обеспечивается выполнение критерия справедливости и критерия живости.

1.3. Описание процессов

В решении используются различные процессы, в совокупности выполняющие функционал светофора, обеспечивающие создание движения и управляющие потоками.

1.3.1. Процесс AddCarsToDirection

Данный процесс обеспечивает создание движения путем установления параметра cars_in_direction в состояниях в значение 1 при отсутствии машин в данном направлении.

```
proctype AddCarsToDirection() {
      do
      :: !directions[0].cars_in_direction -> atomic
{directions[0].cars_in_direction = 1; printf("Added cars to 0 direction: NS\n")}
      :: !directions[1].cars_in_direction -> atomic
{directions[1].cars_in_direction = 1; printf("Added cars to 1 direction: WN\n")}
      :: !directions[2].cars_in_direction -> atomic
{directions[2].cars_in_direction = 1; printf("Added cars to 2 direction: SD\n")}
      :: !directions[3].cars_in_direction -> atomic
{directions[3].cars_in_direction = 1; printf("Added cars to 3 direction: ED\n")}
      :: !directions[4].cars_in_direction -> atomic
{directions[4].cars_in_direction = 1; printf("Added cars to 4 direction: DE\n")}
      :: !directions[5].cars_in_direction -> atomic
{directions[5].cars_in_direction = 1; printf("Added cars to 5 direction: DN\n")}
      od
}
```

1.3.2. Процесс MarkAsShouldBeOpenNext

Процесс обновляет значения should_be_open_next у всех направлений в случае, если каждый из них на этой «итерации» завершил свою работу. Таким образом, между ситуациями, когда ни одно из направлений не должно быть открыто, каждое из направлений выполнит свою работу.

```
proctype MarkAsShouldBeOpenNext() {
    do
    :: !directions[0].should_be_open_next && !directions[1].should_be_open_next
&& !directions[2].should_be_open_next &&  !directions[3].should_be_open_next &&
!directions[4].should_be_open_next && !directions[5].should_be_open_next -> {
           directions[0].should_be_open_next = 1;
directions[1].should_be_open_next = 1; directions[2].should_be_open_next = 1;
           directions[3].should_be_open_next = 1;
directions[4].should_be_open_next = 1; directions[5].should_be_open_next = 1;
       }
    od
}
```

1.3.3. Процесс ControlDirection(int id)

Данный процесс при создании принимает значение идентификатора направления в массиве направлений (`Direction directions[6];`) – 0, 1, 2, 3, 4, 5 для направлений NS, WN, SD, ED, DE и DN соответственно. Процесс управляет состоянием собственного направления.

В бесконечном цикле он выполняет следующие действия (атомарно для одновременного выполнения операций внутри тела условия):

- Если в данном направлении есть машины, направление готово к открытию, направление не открыто и при этом все конфликтующие направления закрыты – направление может быть открыто
- Если в данном направлении есть машины, направление готово к открытию и уже открыто – необходимо пропустить машины в этом направлении и отметить, что оно выполнило свою работу на этой «итерации»
- Если направление выполнило свою работу и при этом открыто – необходимо закрыть это направление
- Если направление готово к открытию, но оно закрыто и машин в направлении нет – считаем, что оно выполнило свою работу на этой «итерации»

```
proctype ControlDirection(int id) {
	do
	// If there are any cars in direction && it should be open && it is not
open && all conflict directions[0] are closed -> set as open
	:: directions[id].cars_in_direction == 1 &&
directions[id].should_be_open_next == 1 && !directions[id].open &&
		((id == 0 && !directions[1].open && !directions[2].open &&
!directions[3].open && !directions[4].open && !directions[5].open) || (id == 1
&& !directions[0].open && !directions[2].open && !directions[3].open &&
!directions[4].open) || (id == 2 && !directions[0].open && !directions[1].open
&& !directions[4].open && !directions[5].open) || (id == 3 &&
!directions[0].open && !directions[1].open && !directions[5].open) || (id == 4
&& !directions[0].open && !directions[1].open && !directions[2].open) || (id ==
5 && !directions[0].open && !directions[2].open && !directions[3].open)) ->
atomic {
```

```
                directions[id].open = ((id == 0 && !directions[1].open &&
!directions[2].open && !directions[3].open && !directions[4].open &&
!directions[5].open) || (id == 1 && !directions[0].open && !directions[2].open
&& !directions[3].open && !directions[4].open) || (id == 2 &&
!directions[0].open && !directions[1].open && !directions[4].open &&
!directions[5].open) || (id == 3 && !directions[0].open && !directions[1].open
&& !directions[5].open) || (id == 4 && !directions[0].open &&
!directions[1].open && !directions[2].open) || (id == 5 && !directions[0].open
&& !directions[2].open && !directions[3].open));
                printf("Opened %d direction\n", id)
            }
      // If there are any cars in direction && it should be open && it is open -
> remove cars and set as should not be open
      :: directions[id].cars_in_direction == 1 &&
       directions[id].should_be_open_next == 1 &&
       directions[id].open == 1 -> atomic {
           directions[id].cars_in_direction = 0;
               directions[id].should_be_open_next = 0;
               printf("Removed cars from %d direction\n", id)
         }
      :: !directions[id].should_be_open_next &&
       directions[id].open == 1 -> atomic {
           directions[id].open = 0;
               printf("Close %d direction after opening\n", id)
         }
      // If direction should be open but there is no cars and it is closed ->
direction should not be open
      :: directions[id].should_be_open_next == 1 &&
       !directions[id].open &&
       !directions[id].cars_in_direction -> atomic {
           directions[id].should_be_open_next = 0;
             printf("Close %d direction because there are no cars inside\n", id)
         }
      od
}
```

1.3.4. Процесс init

Процесс init отвечает за одновременное (atomic) создание всех описанных выше процессов, включая AddCarsToDirection, MarkAsShouldBeOpenNext и шести процессов, отвечающих за направления.

2. ВЕРИФИКАЦИЯ МОДЕЛИ

Для верификации реализованной модели было проверено, что для каждого из направлений NS, WN, SD, ED, DE и DN соблюдаются критерии безопасности, живости и справедливости. Критерии были описаны соответствующими LTL-формулами.

2.1. Свойство безопасности

Формулировка – «Никогда не случится ситуации, что направление открыто для движения и при этом какое-либо из конфликтующих направлений также открыто для движения»

LTL формулы для направлений представлены в таблице 2.

Таблица 2. Свойство безопасности

	LTL-формула
NS	$\overline{G(NS.open \;\&\&\; (WN.open \mid SD.open \mid ED.open \mid\; DE.open \mid DN.open))}$
WN	$\overline{G(WN.open \;\&\&\; (NS.open \mid SD.open \mid ED.open \mid DE.open\;))}$
SD	$\overline{G(SD.open \;\&\&\; (NS.open \mid WN.open \mid\; DE.open \mid DN.open))}$
ED	$\overline{G(ED.open \;\&\&\; (NS.open \mid WN.open \mid DN.open))}$
DE	$\overline{G(DE.open \;\&\&\; (NS.open \mid WN.open \mid SD.open))}$
DN	$\overline{G(DN.open \;\&\&\; (NS.open \mid SD.open \mid ED.open))}$

Описание LTL-формул в Promela:

```
ltl safety0 { [] ! (directions[0].open && (directions[1].open ||
directions[2].open || directions[3].open || directions[4].open ||
directions[5].open))}
ltl safety1 { [] ! (directions[1].open && (directions[0].open ||
directions[2].open || directions[3].open || directions[4].open))}
ltl safety2 { [] ! (directions[2].open && (directions[0].open ||
directions[1].open || directions[4].open || directions[5].open))}
ltl safety3 { [] ! (directions[3].open && (directions[0].open ||
directions[1].open || directions[5].open))}
ltl safety4 { [] ! (directions[4].open && (directions[0].open ||
directions[1].open || directions[2].open))}
ltl safety5 { [] ! (directions[5].open && (directions[0].open ||
directions[2].open || directions[3].open))}
```

2.2. Свойство живости

Формулировка – «Всегда правда, что если в направлении присутствуют машины и направление закрыто – то когда-нибудь направление откроется»

LTL-формула одинакова для всех направлений (DIR - направление):

$$G((DIR.cars \;\&\&\; \overline{DIR.open}) \rightarrow F(DIR.open))$$

```
ltl liveness0 { [] ((directions[0].cars_in_direction && !directions[0].open) ->
<> (directions[0].open))}
ltl liveness1 { [] ((directions[1].cars_in_direction && !directions[1].open) ->
<> (directions[1].open))}
ltl liveness2 { [] ((directions[2].cars_in_direction && !directions[2].open) ->
<> (directions[2].open))}
ltl liveness3 { [] ((directions[3].cars_in_direction && !directions[3].open) ->
<> (directions[3].open))}
ltl liveness4 { [] ((directions[4].cars_in_direction && !directions[4].open) ->
<> (directions[4].open))}
ltl liveness5 { [] ((directions[5].cars_in_direction && !directions[5].open) ->
<> (directions[5].open))}
```

2.3. Свойство справедливости

Формулировка – «Всегда в будущем либо направление закрыто, либо на направлении нет машин»

LTL-формула одинакова для всех направлений (DIR - направление):

$$GF\overline{(DIR.open \;\&\&\; DIR.cars)}$$

```
ltl fairness0 { [] (<> (! (directions[0].open &&
directions[0].cars_in_direction)))}
ltl fairness1 { [] (<> (! (directions[1].open &&
directions[1].cars_in_direction)))}
ltl fairness2 { [] (<> (! (directions[2].open &&
directions[2].cars_in_direction)))}
ltl fairness3 { [] (<> (! (directions[3].open &&
directions[3].cars_in_direction)))}
ltl fairness4 { [] (<> (! (directions[4].open &&
directions[4].cars_in_direction)))}
ltl fairness5 { [] (<> (! (directions[5].open &&
directions[5].cars_in_direction)))}
```

2.4. Верификация в Spin

При верификации модели с помощью Spin были установлены параметры, представленные на рисунке 2.

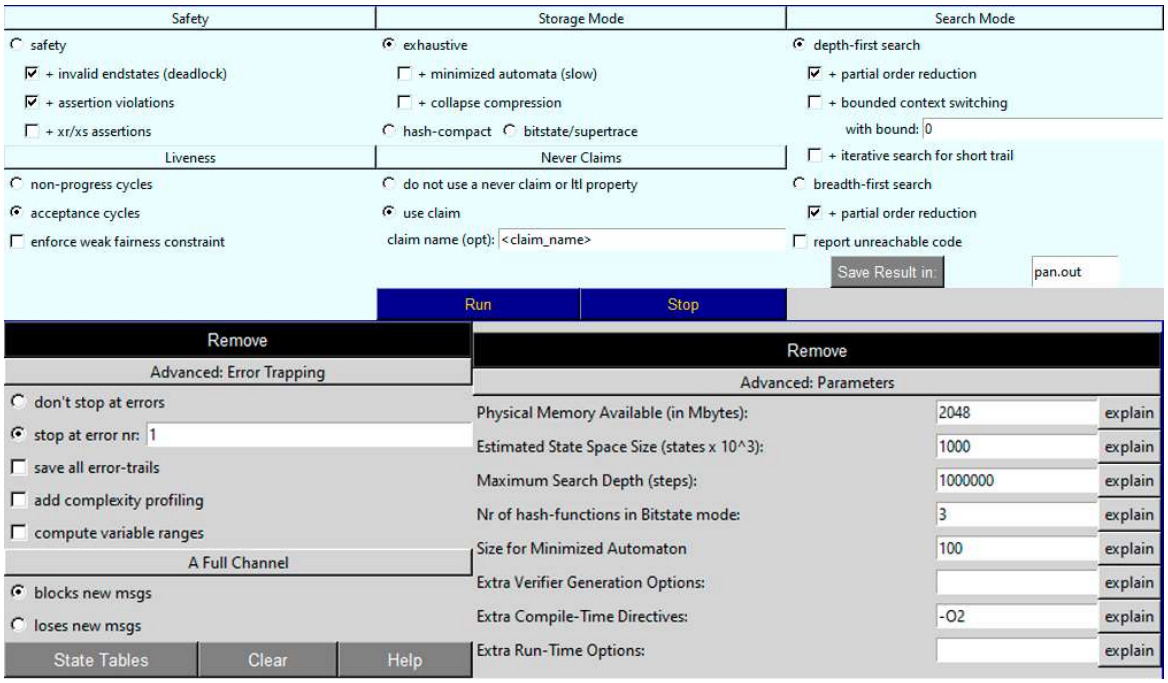

Рисунок 2. Параметры для верификации свойств

Функция «report unreachable code» отключена по причине, что предполагается, что модель работает бесконечно, а значит процессы не приходят в состояние “-end-“.

Результаты верификации на примере направления NS свойств безопасности (safety0), безопасности (liveness0) и справедливости (fairness0) представлены в приложениях B, C и D соответственно.

ЗАКЛЮЧЕНИЕ

В результате выполнения данной работы была выполнена задача моделирования сложного перекрестка, содержащего конфликтующие направления. По итогам ее выполнения была реализована модель перекрестка на языке Promela, использующая 2 процесса, управляющих внешней средой и 6 процессов, управляющих направлениями, заданными в условии задачи.

Для итоговой модели была проведена верификация свойств безопасности, живости и справедливости, выраженных в LTL-формулах. По результатам верификации можно сказать, что модель корректна.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпов Ю.Г., Шошмина И.В. Верификация распределенных систем: учеб. пособие СПб.: Изд-во Политехнического университета, 2011. 212 с.

2. Шошмина И.В., Карпов Ю.Г. Введение в язык Promela и систему комплексной верификации Spin: учеб. пособие СПб.: Изд-во Политехнического университета, 2009

3. Карпов Ю.Г. Model checking. Верификация параллельных и распределенных программных систем. // БХВ-Петербург, 2009, 520 с.

4. Документация к Promela // Promela Manual Pages [Электронный ресурс]. URL: https://spinroot.com/spin/Man/promela.html (дата обращения: 26.04.2024)

5. Документация к Spin // Spin Online References [Электронный ресурс]. URL: https://spinroot.com/spin/Man/ (дата обращения: 26.04.2024)

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД МОДЕЛИ

Название файла - traffic-light.pml

```
typedef Direction {
	bool cars_in_direction = 0;
	bool open = 0;
	bool should_be_open_next = 0;
};
//NS WN SD ED DE DN
Direction directions[6];
// Checks
// Safety: Always (if direction is open then conflicting one is open) is
False
ltl safety0 { [] ! (directions[0].open && (directions[1].open ||
directions[2].open || directions[3].open || directions[4].open ||
directions[5].open))}
ltl safety1 { [] ! (directions[1].open && (directions[0].open ||
directions[2].open || directions[3].open || directions[4].open))}
ltl safety2 { [] ! (directions[2].open && (directions[0].open ||
directions[1].open || directions[4].open || directions[5].open))}
ltl safety3 { [] ! (directions[3].open && (directions[0].open ||
directions[1].open || directions[5].open))}
ltl safety4 { [] ! (directions[4].open && (directions[0].open ||
directions[1].open || directions[2].open))}
ltl safety5 { [] ! (directions[5].open && (directions[0].open ||
directions[2].open || directions[3].open))}
// Liveness: Always (if there are cars in direction and it is closed then
later it will open) is True
ltl liveness0 { [] ((directions[0].cars_in_direction &&
!directions[0].open) -> <> (directions[0].open))}
ltl liveness1 { [] ((directions[1].cars_in_direction &&
!directions[1].open) -> <> (directions[1].open))}
ltl liveness2 { [] ((directions[2].cars_in_direction &&
!directions[2].open) -> <> (directions[2].open))}
ltl liveness3 { [] ((directions[3].cars_in_direction &&
!directions[3].open) -> <> (directions[3].open))}
ltl liveness4 { [] ((directions[4].cars_in_direction &&
!directions[4].open) -> <> (directions[4].open))}
```

```
ltl liveness5 { [] ((directions[5].cars_in_direction &&
!directions[5].open) -> <> (directions[5].open))}
// Fairness: Always in the future either there won't be any cars either
the direction will close
ltl fairness0 { [] (<> (! (directions[0].open &&
directions[0].cars_in_direction)))}
ltl fairness1 { [] (<> (! (directions[1].open &&
directions[1].cars_in_direction)))}
ltl fairness2 { [] (<> (! (directions[2].open &&
directions[2].cars_in_direction)))}
ltl fairness3 { [] (<> (! (directions[3].open &&
directions[3].cars_in_direction)))}
ltl fairness4 { [] (<> (! (directions[4].open &&
directions[4].cars_in_direction)))}
ltl fairness5 { [] (<> (! (directions[5].open &&
directions[5].cars_in_direction)))}
// Process that adds new cards to any empty direction
proctype AddCarsToDirection() {
     do
     :: !directions[0].cars_in_direction -> atomic
{directions[0].cars_in_direction = 1; printf("Added cars to 0 direction:
NS\n")}
     :: !directions[1].cars_in_direction -> atomic
{directions[1].cars_in_direction = 1; printf("Added cars to 1 direction:
WN\n")}
     :: !directions[2].cars_in_direction -> atomic
{directions[2].cars_in_direction = 1; printf("Added cars to 2 direction:
SD\n")}
     :: !directions[3].cars_in_direction -> atomic
{directions[3].cars_in_direction = 1; printf("Added cars to 3 direction:
ED\n")}
     :: !directions[4].cars_in_direction -> atomic
{directions[4].cars_in_direction = 1; printf("Added cars to 4 direction:
DE\n")}
     :: !directions[5].cars_in_direction -> atomic
{directions[5].cars_in_direction = 1; printf("Added cars to 5 direction:
DN\n")}
     od
```

```
}
// Process that marks all the directions[0] as should_be_open_next
proctype MarkAsShouldBeOpenNext() {
    do
    :: !directions[0].should_be_open_next &&
!directions[1].should_be_open_next && !directions[2].should_be_open_next
&&
     !directions[3].should_be_open_next &&
!directions[4].should_be_open_next && !directions[5].should_be_open_next
-> {
           directions[0].should_be_open_next = 1;
directions[1].should_be_open_next = 1; directions[2].should_be_open_next
= 1;
           directions[3].should_be_open_next = 1;
directions[4].should_be_open_next = 1; directions[5].should_be_open_next
= 1;
       }
    od
}
proctype ControlDirection(int id) {
     do
     // If there are any cars in direction && it should be open && it is
not open && all conflict directions[0] are closed -> set as open
     :: directions[id].cars_in_direction == 1 &&
directions[id].should_be_open_next == 1 && !directions[id].open &&
           ((id == 0 && !directions[1].open && !directions[2].open &&
!directions[3].open && !directions[4].open && !directions[5].open) ||
           (id == 1 && !directions[0].open && !directions[2].open &&
!directions[3].open && !directions[4].open) ||
           (id == 2 && !directions[0].open && !directions[1].open &&
!directions[4].open && !directions[5].open) ||
           (id == 3 && !directions[0].open && !directions[1].open &&
!directions[5].open) ||
           (id == 4 && !directions[0].open && !directions[1].open &&
!directions[2].open) ||
           (id == 5 && !directions[0].open && !directions[2].open &&
!directions[3].open)) -> atomic {
```

```
                directions[id].open = ((id == 0 && !directions[1].open
&& !directions[2].open && !directions[3].open && !directions[4].open &&
!directions[5].open) ||
                    (id == 1 && !directions[0].open &&
!directions[2].open && !directions[3].open && !directions[4].open) ||
                    (id == 2 && !directions[0].open &&
!directions[1].open && !directions[4].open && !directions[5].open) ||
                    (id == 3 && !directions[0].open &&
!directions[1].open && !directions[5].open) ||
                    (id == 4 && !directions[0].open &&
!directions[1].open && !directions[2].open) ||
                    (id == 5 && !directions[0].open &&
!directions[2].open && !directions[3].open));
                printf("Opened %d direction\n", id)
          }
     // If there are any cars in direction && it should be open && it is
open -> remove cars and set as should not be open
     :: directions[id].cars_in_direction == 1 &&
       directions[id].should_be_open_next == 1 &&
       directions[id].open == 1 -> atomic {
           directions[id].cars_in_direction = 0;
              directions[id].should_be_open_next = 0;
              printf("Removed cars from %d direction\n", id)
        }
     // If direction should not be open but open -> close it
     :: !directions[id].should_be_open_next &&
       directions[id].open == 1 -> atomic {
           directions[id].open = 0;
              printf("Close %d direction after opening\n", id)
        }
     // If direction should be open but there is no cars and it is
closed -> direction should not be open
     :: directions[id].should_be_open_next == 1 &&
       !directions[id].open &&
       !directions[id].cars_in_direction -> atomic {
           directions[id].should_be_open_next = 0;
              printf("Close %d direction because there are no cars
inside\n", id)
```

```
        }
    od
}
init {
    atomic {
        byte i;
        for (i : 0..5){
            run ControlDirection(i);
        }

        run AddCarsToDirection();
        run MarkAsShouldBeOpenNext();
    }
}
```

ПРИЛОЖЕНИЕ В

ВЕРИФИКАЦИЯ СВОЙСТВА БЕЗОПАСНОСТИ (NS)

```
./pan -m1000000  -a -n -c1 -N safety0
Pid: 14924
pan: ltl formula safety0
Depth=  338172 States=    1e+06 Transitions= 4.28e+06 Memory=   304.062
     t=    0.972 R=   1e+06
Depth=  455436 States=    2e+06 Transitions= 9.44e+06 Memory=   426.328
     t=     2.12 R=   9e+05
Depth=  522384 States=    3e+06 Transitions= 1.48e+07 Memory=   548.496
     t=     3.27 R=   9e+05
Depth=  581524 States=    4e+06 Transitions= 2.04e+07 Memory=   670.664
     t=      4.5 R=   9e+05
Depth=  598622 States=    5e+06 Transitions= 2.62e+07 Memory=   792.734
     t=     5.72 R=   9e+05
Depth=  614482 States=    6e+06 Transitions= 3.24e+07 Memory=   914.804
     t=     7.06 R=   9e+05
Depth=  614488 States=    7e+06 Transitions= 3.85e+07 Memory=  1036.875
     t=     8.42 R=   8e+05

(Spin Version 6.5.1 -- 20 December 2019)
     + Partial Order Reduction

Full statespace search for:
     never claim          + (safety0)
     assertion violations  + (if within scope of claim)
     acceptance   cycles   + (fairness disabled)
     invalid end states    - (disabled by never claim)

State-vector 120 byte, depth reached 614488, errors: 0
  7980433 states, stored
 36558913 states, matched
 44539346 transitions (= stored+matched)
       21 atomic steps
hash conflicts:   5149518 (resolved)

Stats on memory usage (in Megabytes):
 1065.503  equivalent memory usage for states (stored*(State-vector +
overhead))
  975.452  actual memory usage for states (compression: 91.55%)
           state-vector as stored = 108 byte + 20 byte overhead
  128.000  memory used for hash table (-w24)
   53.406  memory used for DFS stack (-m1000000)
 1156.601  total actual memory usage



pan: elapsed time 9.9 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ С

ВЕРИФИКАЦИЯ СВОЙСТВА ЖИВОСТИ (NS)

```
./pan -m1000000  -a -n -c1 -N liveness0
Pid: 21264
pan: ltl formula liveness0
Depth=     156 States=    1e+06 Transitions= 5.36e+06 Memory=   242.636
      t=    0.805 R=   1e+06
Depth=     336 States=    2e+06 Transitions= 1.12e+07 Memory=   303.672
      t=     1.67 R=   1e+06
Depth=     336 States=    3e+06 Transitions= 1.67e+07 Memory=   364.707
      t=     2.52 R=   1e+06
Depth=  146782 States=    4e+06 Transitions= 2.21e+07 Memory=   437.070
      t=     3.47 R=   1e+06
Depth=  289962 States=    5e+06 Transitions= 2.85e+07 Memory=   529.843
      t=      4.9 R=   1e+06
Depth=  414900 States=    6e+06 Transitions= 3.56e+07 Memory=   636.875
      t=      6.6 R=   9e+05
Depth=  462198 States=    7e+06 Transitions=  4.3e+07 Memory=   745.371
      t=     8.22 R=   9e+05
Depth=  520480 States=    8e+06 Transitions= 5.03e+07 Memory=   850.742
      t=     9.76 R=   8e+05
Depth=  562420 States=    9e+06 Transitions= 5.77e+07 Memory=   954.746
      t=     11.3 R=   8e+05
Depth=  590104 States=    1e+07 Transitions= 6.51e+07 Memory=  1060.996
      t=     12.8 R=   8e+05
Depth=  602256 States=  1.1e+07 Transitions= 7.26e+07 Memory=  1165.976
      t=     14.4 R=   8e+05
Depth=  614478 States=  1.2e+07 Transitions= 8.08e+07 Memory=  1265.195
      t=     16.1 R=   7e+05
Depth=  614488 States=  1.3e+07 Transitions=  8.9e+07 Memory=  1372.812
      t=     17.9 R=   7e+05
Depth=  614488 States=  1.4e+07 Transitions= 9.66e+07 Memory=  1477.011
      t=     19.6 R=   7e+05
Depth=  614488 States=  1.5e+07 Transitions= 1.04e+08 Memory=  1579.648
      t=     21.2 R=   7e+05

(Spin Version 6.5.1 -- 20 December 2019)
      + Partial Order Reduction

Full statespace search for:
      never claim          + (liveness0)
      assertion violations  + (if within scope of claim)
      acceptance   cycles   + (fairness disabled)
      invalid end states    - (disabled by never claim)

State-vector 120 byte, depth reached 614488, errors: 0
 11534190 states, stored (1.50879e+07 visited)
 89548313 states, matched
1.0463626e+08 transitions (= visited+matched)
       21 atomic steps
hash conflicts:  18239880 (resolved)

Stats on memory usage (in Megabytes):
 1539.981  equivalent memory usage for states (stored*(State-vector +
overhead))
 1409.255  actual memory usage for states (compression: 91.51%)
```

```
          state-vector as stored = 108 byte + 20 byte overhead
  128.000  memory used for hash table (-w24)
   53.406  memory used for DFS stack (-m1000000)
 1590.390  total actual memory usage


pan: elapsed time 21.4 seconds
No errors found -- did you verify all claims?
```

ПРИЛОЖЕНИЕ D

ВЕРИФИКАЦИЯ СВОЙСТВА СПРАВЕДЛИВОСТИ (NS)

```
./pan -m1000000  -a -n -c1 -N fairness0
Pid: 22116
pan: ltl formula fairness0
Depth=  273914 States=    1e+06 Transitions=  3.9e+06 Memory=   280.332
      t=    0.829 R=   1e+06
Depth=  408226 States=    2e+06 Transitions= 8.78e+06 Memory=   388.925
      t=     1.85 R=   1e+06
Depth=  467050 States=    3e+06 Transitions= 1.43e+07 Memory=   505.234
      t=        3 R=   1e+06
Depth=  537742 States=    4e+06 Transitions= 1.97e+07 Memory=   623.300
      t=     4.14 R=   1e+06
Depth=  587012 States=    5e+06 Transitions= 2.53e+07 Memory=   741.855
      t=     5.36 R=   9e+05
Depth=  601638 States=    6e+06 Transitions= 3.13e+07 Memory=   859.629
      t=     6.59 R=   9e+05
Depth=  614482 States=    7e+06 Transitions= 3.75e+07 Memory=   977.402
      t=     7.95 R=   9e+05
Depth=  614488 States=    8e+06 Transitions= 4.37e+07 Memory=  1096.347
      t=     9.32 R=   9e+05
Depth=  614488 States=    9e+06 Transitions= 4.98e+07 Memory=  1216.465
      t=     10.7 R=   8e+05

(Spin Version 6.5.1 -- 20 December 2019)
      + Partial Order Reduction

Full statespace search for:
      never claim          + (fairness0)
      assertion violations  + (if within scope of claim)
      acceptance   cycles   + (fairness disabled)
      invalid end states    - (disabled by never claim)

State-vector 120 byte, depth reached 614488, errors: 0
  8509704 states, stored (9.03898e+06 visited)
 41250330 states, matched
 50289305 transitions (= visited+matched)
       21 atomic steps
hash conflicts:   6141511 (resolved)

Stats on memory usage (in Megabytes):
 1136.168  equivalent memory usage for states (stored*(State-vector +
overhead))
 1040.116  actual memory usage for states (compression: 91.55%)
           state-vector as stored = 108 byte + 20 byte overhead
  128.000  memory used for hash table (-w24)
   53.406  memory used for DFS stack (-m1000000)
 1221.250  total actual memory usage



pan: elapsed time 10.9 seconds
No errors found -- did you verify all claims?
```