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

КУРСОВАЯ РАБОТА

по дисциплине «Верификация распределенных алгоритмов»

Тема: Разработка контроллера светофоров и его верификация

Студент гр. 9303 _____ Колованов Р.А.

Преподаватель _____ Шошмина И.В.

Санкт-Петербург

2024

ЗАДАНИЕ

НА КУРСОВУЮ РАБОТУ

Студент Колованов Р.А.

Группа 9303

Тема работы: Разработка контроллера светофоров и его верификация

Исходные данные:

- Выдается несколько пересечений на сложном перекрестке;
- Необходимо составить модель контроллера светофора, управляющего движением автомобилей по перекрестку:
  - 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;
- Верифицировать ее относительно заданных требований.

Содержание пояснительной записки:

«Содержание», «Введение», «Ход работа», «Заключение», «Список использованных источников»

Предполагаемый объем пояснительной записки:

Не менее 10 страниц.

Дата выдачи задания: 11.04.2024

Дата сдачи реферата: 06.06.2024

Дата защиты реферата: 06.06.2024

Студент _____ Колованов Р.А.

Преподаватель _____ Шошмина И.В.

АННОТАЦИЯ

В данной курсовой работе рассматривается задача разработки контроллера светофора для управления движением автомобилей на сложном перекрестке. Была разработана модель контроллера светофоров на языке Promela. Разработанная модель была верифицирована по трем свойствам: безопасность, живость и справедливость.

SUMMARY

In this course work, the task of developing a traffic light controller for controlling the movement of cars at a complex intersection is considered. A traffic lights controller model has been developed in the Promela language. The developed model was verified according to three properties: safety, liveliness and fairness.

СОДЕРЖАНИЕ

	Введение	5
1.	Разработка модели	6
1.1.	Задание	6
1.2.	Описание состояний модели	7
1.3.	Описание процессов модели	7
2.	Верификация модели	9
2.1.	Верификация свойства безопасности	9
2.2.	Верификация свойства живости	9
2.3.	Верификация свойства справедливости	10
	Заключение	11
	Список использованных источников	12
	Приложение А. Модель на языке Promela	13

ВВЕДЕНИЕ

Проверка корректности работы распределенных алгоритмов зачастую является крайне трудоемкой задачей, в которой требуется рассмотреть все возможные состояния системы. Для решения данной задачи были разработаны специальные инструменты – верификаторы, которые на основании разработанной модели системы проверяют выполнимость заданных требований. В данной работе рассматривается язык Promela для моделирования распределенных систем и верификатор Spin.

Целью работы является разработка модели контроллера светофоров на перекрестке, которая будет обрабатывать недетерминированный поток машин в соответствии с заданными свойствами. Для разработки и верификации модели распределенного алгоритма контроллера светофоров необходимо обойтись без предположений об очередности действий и дать возможность контроллерам движения и контроллерам светофоров действовать независимо друг от друга в борьбе за общие ресурсы. Также необходимо учесть, что перекресток могут пересекать несколько потоков машин одновременно, если направления движения не пересекаются.

1. РАЗРАБОТКА МОДЕЛИ

1.1. Задание

Вариант 5. Пересечения SD/WN, SD/DN, WN/DE, NS/WE (24, 34, 43, 5).

Суммарно на сложном перекрестке имеется 6 направлений движения (SD, WN, DN, DE, NS, WE), которые образуют 10 пересечений друг с другом.

Схема сложного перекрестка с направлениями движения и точками пересечения представлена на рис. 1.

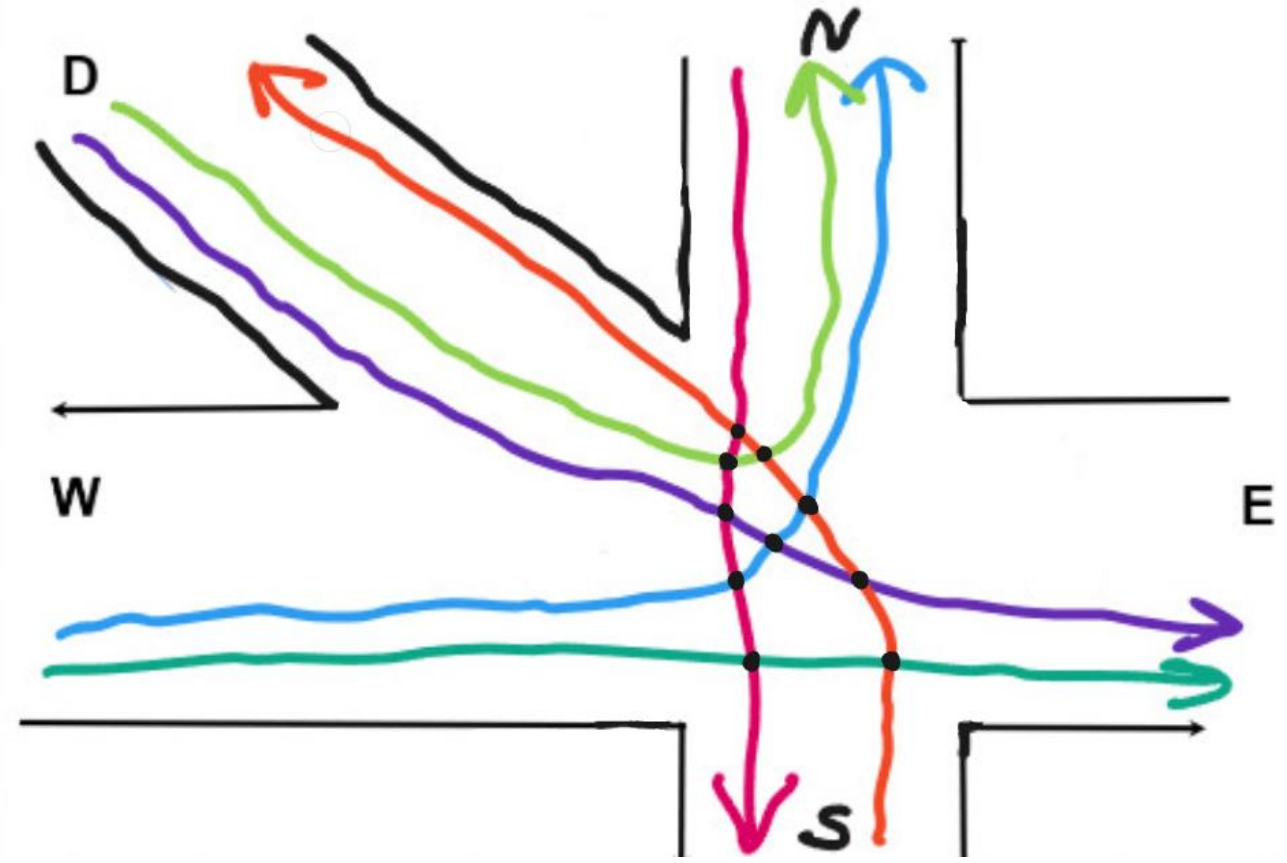

Рисунок 1 – Схема сложного перекрестка с направлениями движения и точками пересечения

Направление SD пересекается с направлениями WE, DE, WN, DN, NS; направление WN – с NS, DE, SD; направление DN – с NS, SD; направление DE – с NS, WN, SD; направление NS – с SD, DN, DE, WN, WE; направление WE – с NS, SD.

1.2. Описание состояний модели

Для разработки модели необходимо выделить переменные, которые будут формировать состояния модели, то есть определять текущую ситуацию на перекрестке. Состояние каждого из шести направлений сложного перекрестка было представлено в виде двух переменных:

- Переменная «XX_SENSE», где XX – направление. Является логической переменной и хранит информацию о наличии потока машин в соответствующем направлении;
- Переменная «XX_LIGHT», где XX – направление. Является целочисленной переменной и хранит информацию о текущем цвете света светофора. Символьная константа RED означает, что горит красный свет, символьная константа GREEN – что зеленый свет.

Для синхронизации процессов были введены логические переменные «XX_LOCK», где XX – направление. Если «XX_LOCK» равен true, то процесс контроллера направления XX захватил ресурс и может пускать машины по своему направлению через перекресток.

1.3. Описание процессов модели

Для модели были разработаны шесть процессов ControllerXX, которые представляют собой независимые контроллеры светофора на соответствующих направлениях перекрестка (XX – направление), и процесс EnvironmentController, который представляет собой внешнюю среду. В завершение был написан главный процесс init, осуществляющий запуск процессов контроллеров и процесса внешней среды атомарно.

Процесс EnvironmentController представляет собой бесконечный цикл, который отвечает за работу датчиков наличия машины на всех направлениях (XX_SENSE). Если на направлении нет машин и включен красный свет светофора, то на направлении может появиться поток машин. Если же на направлении есть поток машин и горит зеленый свет светофора (т.е. машины в

данный момент движутся через перекресток), то на направлении поток машин может закончиться.

Все процессы контроллеров светофоров ControllerXX схожи по реализации. В первом условии они проверяют, есть ли запрос на проезд в их направлении и не горит ли уже зеленый свет светофора. Если это так, то атомарно делается проверка на отсутствие блокировок на конфликтующих направлениях, после чего устанавливается блокировка и включает зеленый свет. Считается, что в этот момент машины начинают проезжать в направлении. Вторым условием является наличие включенного зеленого света светофора и отсутствие запроса на проезд. Если оно выполнено, сначала выключается свет, а затем снимается блокировка. Порядок важен для безопасности.

Полный код модели на Promela представлен в Приложении А.

2. ВЕРИФИКАЦИЯ МОДЕЛИ

Для верификации модели было проверено выполнение свойств безопасности, живости и справедливости для каждого направления.

2.1. Верификация свойства безопасности

Свойство безопасности означает, что никогда не будет разрешен проезд в пересекающихся направлениях.

Свойство безопасности для направлений были представлены следующими формулами:

```
safetySD ::= G !((SD_LIGHT == GREEN) && (WE_LIGHT == GREEN || DE_LIGHT == GREEN || WN_LIGHT == GREEN || DN_LIGHT == GREEN || NS_LIGHT == GREEN))
safetyWN ::= G !((WN_LIGHT == GREEN) && (DE_LIGHT == GREEN || SD_LIGHT == GREEN || NS_LIGHT == GREEN))
safetyDN ::= G !((DN_LIGHT == GREEN) && (SD_LIGHT == GREEN || NS_LIGHT == GREEN))
safetyDE ::= G !((DE_LIGHT == GREEN) && (SD_LIGHT == GREEN || WN_LIGHT == GREEN || NS_LIGHT == GREEN))
safetyNS ::= G !((NS_LIGHT == GREEN) && (WE_LIGHT == GREEN || DE_LIGHT == GREEN || WN_LIGHT == GREEN || DN_LIGHT == GREEN || SD_LIGHT == GREEN))
safetyWE ::= G !((WE_LIGHT == GREEN) && (SD_LIGHT == GREEN || NS_LIGHT == GREEN))
```

Результаты верификации подтвердили выполнение свойства безопасности для каждого направления.

2.2. Верификация свойства живости

Свойство живости означает, что при появлении машины ей всегда предоставится возможность проезда в нужном направлении (возможно, не сразу).

Свойство живости для направлений были представлены следующими формулами:

```
livenessSD ::= G((SD_SENSE && SD_LIGHT == RED) -> F(SD_LIGHT == GREEN))
livenessWN ::= G ((WN_SENSE && WN_LIGHT == RED) -> F(WN_LIGHT == GREEN))
livenessDN ::= G ((DN_SENSE && DN_LIGHT == RED) -> F(DN_LIGHT == GREEN))
livenessDE ::= G ((DE_SENSE && DE_LIGHT == RED) -> F(DE_LIGHT == GREEN))
livenessNS ::= G ((NS_SENSE && NS_LIGHT == RED) -> F(NS_LIGHT == GREEN))
livenessWE ::= G ((WE_SENSE && WE_LIGHT == RED) -> F(WE_LIGHT == GREEN))
```

Результаты верификации не подтвердили выполнение свойства живости для какого-либо направления. Были найдены контрпримеры. Это можно объяснить наличием голодания.

2.3. Верификация свойства справедливости

Свойство справедливости означает, что на каждом направлении не движется непрерывный поток машин.

Свойство справедливости для направлений были представлены следующими формулами:

```
fairnessSD ::= GF !(SD_LIGHT == GREEN && SD_SENSE)
fairnessWN ::= GF !(WN_LIGHT == GREEN && WN_SENSE)
fairnessDN ::= GF !(DN_LIGHT == GREEN && DN_SENSE)
fairnessDE ::= GF !(DE_LIGHT == GREEN && DE_SENSE)
fairnessNS ::= GF !(NS_LIGHT == GREEN && NS_SENSE)
fairnessWE ::= GF !(WE_LIGHT == GREEN && WE_SENSE)
```

Результаты верификации не подтвердили выполнение свойства справедливости для некоторых направлений. Были найдены контрпримеры. Это можно объяснить наличием голодания.

ЗАКЛЮЧЕНИЕ

В результате выполнения работы была разработана модель контроллера светофоров на сложном перекрестке. Взятый за основу алгоритм из методических указаний для простого перекрестка был доработан. В модели использована общая память процессов для хранения состояний контроллеров, продуманы ограничения системы для соблюдения свойства безопасности. Выполнена верификация модели по критериям безопасности, живности и справедливости, выраженным в LTL-формулах.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Карпов Ю. Г., Шошмина И. В. Верификация распределенных систем: учебное пособие для студентов высших учебных заведений, обучающихся по направлению подготовки магистров в образовательной области «Информатика и вычислительная техника». – 2011.

2. SPIN Verifier’s Roadmap: Spin // spinroot.com. URL: https://spinroot.com/spin/Man/4_SpinVerification.html (дата обращения: 04.06.2024).

3. Concise Promela Reference // spinroot.com. URL: http://spinroot.com/spin/Man/Quick.html (дата обращения: 04.06.2024).

4. Basic Spin Manual // spinroot.com. URL: https://spinroot.com/spin/Man/Manual.html (дата обращения: 04.06.2024).

ПРИЛОЖЕНИЕ А
МОДЕЛЬ НА ЯЗЫКЕ PROMELA

```
// LTL

ltl safetySD { [] !((SD_LIGHT == GREEN) && (WE_LIGHT == GREEN || DE_LIGHT
== GREEN || WN_LIGHT == GREEN || DN_LIGHT == GREEN || NS_LIGHT == GREEN))
};
ltl safetyWN { [] !((WN_LIGHT == GREEN) && (DE_LIGHT == GREEN || SD_LIGHT
== GREEN || NS_LIGHT == GREEN)) };
ltl safetyDN { [] !((DN_LIGHT == GREEN) && (SD_LIGHT == GREEN || NS_LIGHT
== GREEN)) };
ltl safetyDE { [] !((DE_LIGHT == GREEN) && (SD_LIGHT == GREEN || WN_LIGHT
== GREEN || NS_LIGHT == GREEN)) };
ltl safetyNS { [] !((NS_LIGHT == GREEN) && (WE_LIGHT == GREEN || DE_LIGHT
== GREEN || WN_LIGHT == GREEN || DN_LIGHT == GREEN || SD_LIGHT == GREEN))
};
ltl safetyWE { [] !((WE_LIGHT == GREEN) && (SD_LIGHT == GREEN || NS_LIGHT
== GREEN)) };

ltl livenessSD { [] ((SD_SENSE && SD_LIGHT == RED) -> <> (SD_LIGHT ==
GREEN)) };
ltl livenessWN { [] ((WN_SENSE && WN_LIGHT == RED) -> <> (WN_LIGHT ==
GREEN)) };
ltl livenessDN { [] ((DN_SENSE && DN_LIGHT == RED) -> <> (DN_LIGHT ==
GREEN)) };
ltl livenessDE { [] ((DE_SENSE && DE_LIGHT == RED) -> <> (DE_LIGHT ==
GREEN)) };
ltl livenessNS { [] ((NS_SENSE && NS_LIGHT == RED) -> <> (NS_LIGHT ==
GREEN)) };
ltl livenessWE { [] ((WE_SENSE && WE_LIGHT == RED) -> <> (WE_LIGHT ==
GREEN)) };

ltl fairnessSD { [] <> !(SD_LIGHT == GREEN && SD_SENSE) };
ltl fairnessWN { [] <> !(WN_LIGHT == GREEN && WN_SENSE) };
ltl fairnessDN { [] <> !(DN_LIGHT == GREEN && DN_SENSE) };
ltl fairnessDE { [] <> !(DE_LIGHT == GREEN && DE_SENSE) };
ltl fairnessNS { [] <> !(NS_LIGHT == GREEN && NS_SENSE) };
```

```
ltl fairnessWE { [] <> !(WE_LIGHT == GREEN && WE_SENSE) };

// Traffic lights for direaction: SD, WN, DN, DE, NS, WE
mtype:light = {RED, GREEN};
mtype:light SD_LIGHT = RED;
mtype:light WN_LIGHT = RED;
mtype:light DN_LIGHT = RED;
mtype:light DE_LIGHT = RED;
mtype:light NS_LIGHT = RED;
mtype:light WE_LIGHT = RED;

// Presence of cars in a given direction: SD, WN, DN, DE, NS, WE
bool SD_SENSE = false;
bool WN_SENSE = false;
bool DN_SENSE = false;
bool DE_SENSE = false;
bool NS_SENSE = false;
bool WE_SENSE = false;

// Locks for direction: SD, WN, DN, DE, NS, WE
bool SD_LOCK = false;
bool WN_LOCK = false;
bool DN_LOCK = false;
bool DE_LOCK = false;
bool NS_LOCK = false;
bool WE_LOCK = false;

// Direction controllers: SD, WN, DN, DE, NS, WE
proctype ControllerSD() {
     do
     :: (SD_SENSE && SD_LIGHT == RED) -> {
       atomic { (!WE_LOCK && !DE_LOCK && !WN_LOCK && !DN_LOCK &&
!NS_LOCK); SD_LOCK = true; };
       SD_LIGHT = GREEN;
     };
     :: (!SD_SENSE && SD_LIGHT == GREEN) -> {
          SD_LIGHT = RED;
          SD_LOCK = false;
```

```
      };
      od;
}

proctype ControllerWN() {
      do
      :: (WN_SENSE && WN_LIGHT == RED) ->
        atomic { (!NS_LOCK && !DE_LOCK && !SD_LOCK); WN_LOCK = true; };
           WN_LIGHT = GREEN;
      :: (!WN_SENSE && WN_LIGHT == GREEN) -> {
           WN_LIGHT = RED;
           WN_LOCK = false;
      };
      od;
}

proctype ControllerDN() {
      do
      :: (DN_SENSE && DN_LIGHT == RED) ->
        atomic { (!NS_LOCK && !SD_LOCK); DN_LOCK = true; };
           DN_LIGHT = GREEN;
      :: (!DN_SENSE && DN_LIGHT == GREEN) -> {
           DN_LIGHT = RED;
           DN_LOCK = false;
      };
      od;
}

proctype ControllerDE() {
      do
      :: (DE_SENSE && DE_LIGHT == RED) ->
        atomic { (!NS_LOCK && !WN_LOCK && !SD_LOCK); DE_LOCK = true; };
        DE_LIGHT = GREEN;
      :: (!DE_SENSE && DE_LIGHT == GREEN) -> {
           DE_LIGHT = RED;
           DE_LOCK = false;
      };
      od;
```

```
}

proctype ControllerNS() {
      do
      :: (NS_SENSE && NS_LIGHT == RED) ->
        atomic { (!SD_LOCK && !DN_LOCK && !DE_LOCK && !WN_LOCK &&
!WE_LOCK); NS_LOCK = true; };
           NS_LIGHT = GREEN;
      :: (!NS_SENSE && NS_LIGHT == GREEN) -> {
           NS_LIGHT = RED;
           NS_LOCK = false;
      };
      od;
}

proctype ControllerWE() {
      do
      :: (WE_SENSE && WE_LIGHT == RED) ->
        atomic { (!SD_LOCK && !NS_LOCK); WE_LOCK = true; };
        WE_LIGHT = GREEN;
      :: (!WE_SENSE && WE_LIGHT == GREEN) -> {
           WE_LIGHT = RED;
           WE_LOCK = false;
      };
      od;
}

// External environment controller
proctype EnvironmentController() {
      do
      :: (!SD_SENSE && SD_LIGHT == RED) -> SD_SENSE = true;
      :: (!WN_SENSE && WN_LIGHT == RED) -> WN_SENSE = true;
      :: (!DN_SENSE && DN_LIGHT == RED) -> DN_SENSE = true;
      :: (!DE_SENSE && DE_LIGHT == RED) -> DE_SENSE = true;
      :: (!NS_SENSE && NS_LIGHT == RED) -> NS_SENSE = true;
      :: (!WE_SENSE && WE_LIGHT == RED) -> WE_SENSE = true;

      :: (SD_SENSE && SD_LIGHT == GREEN) -> SD_SENSE = false;
```

```
      :: (WN_SENSE && WN_LIGHT == GREEN) -> WN_SENSE = false;
      :: (DN_SENSE && DN_LIGHT == GREEN) -> DN_SENSE = false;
      :: (DE_SENSE && DE_LIGHT == GREEN) -> DE_SENSE = false;
      :: (NS_SENSE && NS_LIGHT == GREEN) -> NS_SENSE = false;
      :: (WE_SENSE && WE_LIGHT == GREEN) -> WE_SENSE = false;
      od;
}

init {
      atomic {
            run EnvironmentController();
            run ControllerSD();
            run ControllerWN();
            run ControllerDN();
            run ControllerDE();
            run ControllerNS();
            run ControllerWE();
      }
}
```