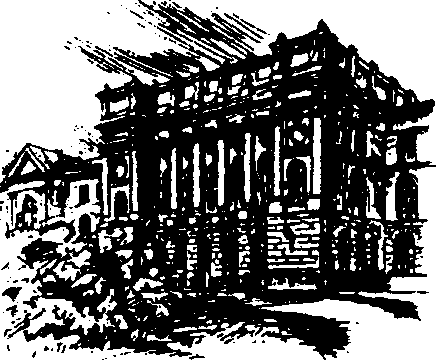
Минобрнауки России

Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

**Кафедра «Информационные и управляющие системы»**



**КУРСОВОЙ ПРОЕКТ**

**Верификация распределенной системы при помощи системы Spin**

по дисциплине «Распределенные алгоритмы и протоколы»

Выполнил

студент гр. 6084/12 А.Лукашин

Руководитель

ст. преподаватель Шошмина И.В.

«\_\_» \_\_\_\_\_\_\_\_\_\_ 2013 г.

Санкт-Петербург

2013

# Введение

Верификация – это проверка модели (алгоритма, системы) на корректность работы, с учетом заданных правил. В настоящее время вопрос корректности работы программных продуктов является крайне актуальным. На программное обеспечивание возлагаются все новые и новые функции, при это в некоторых областях возникновение ошибки может приводить к катастрофическим последствиям. Такие тенденции позволяют говорить о важности применения и развития верификационных средств.

В данной курсовой работе рассматривается модель управления движением на перекретске, при условии,что каждое направление контролируется своим светофором. При этом для model checking используется система Spin. Данная система позволяет описывать модель на языке Promela и задавать требуемые свойства системы при помощи выражений LTL. Такой подход позволяет проверить корректность модели перед ее реализацией.

# Постановка задачи

Вариант (1, 12, 15).

Пересечения: {(NS, WN), (NE, EW),(SW,ES)}

Для заданного враианта необходимо создать набор свойств (LTL), которые определяют корректность работы модели (например отсутствие варианта, когда движение расзрешено всем), а так же описатьсаму модель на языке Promela.

# Реализация

## Модель на языке Promela

Данная модель была построена на основе следующих постулатов:

1. Каждый контроллер светофора является отдельным процессом.
2. Сигнал светофора может быть двух видов: красный – зеленый
3. Состояние светофора описывается глобальными переменными
4. Датчики движения также являются глобальными переменными
5. Движение (траффик машин) генерируется внешним, по отношению к контроллерам, процессом случайным образом

Реализация:

/\* 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;

/\*Safety\*/

ltl p {[] !((NS\_L==Green && WN\_L==Green && SW\_L==Green && EW\_L==Green)

||(WN\_L==Green && NS\_L==Green && NE\_L==Green && SW\_L==Green && EW\_L==Green)

||(EW\_L==Green && NE\_L==Green && WN\_L==Green && NS\_L==Green)

||(NE\_L==Green && WN\_L==Green && ES\_L==Green && EW\_L==Green)

||(SW\_L==Green && ES\_L==Green && WN\_L==Green && NS\_L==Green)

||(ES\_L==Green && SW\_L==Green && NE\_L==Green))}

/\*Liveness\*/

ltl p1\_1 {[] ((NS\_S && (NS\_L==Red)) -> <> (NS\_L==Green))}

ltl p1\_2 {[] ((WN\_S && (WN\_L==Red)) -> <> (WN\_L==Green))}

ltl p1\_3 {[] ((NE\_S && (NE\_L==Red)) -> <> (NE\_L==Green))}

ltl p1\_4 {[] ((EW\_S && (EW\_L==Red)) -> <> (EW\_L==Green))}

ltl p1\_5 {[] ((ES\_S && (ES\_L==Red)) -> <> (ES\_L==Green))}

ltl p1\_6 {[] ((SW\_S && (SW\_L==Red)) -> <> (SW\_L==Green))}

/\*Fairness\*/

ltl p2\_1 {[] <> !((NS\_L==Green) && NS\_S)}

ltl p2\_2 {[] <> !((WN\_L==Green) && WN\_S)}

ltl p2\_3 {[] <> !((NE\_L==Green) && NE\_S)}

ltl p2\_4 {[] <> !((EW\_L==Green) && EW\_S)}

ltl p2\_5 {[] <> !((ES\_L==Green) && ES\_S)}

ltl p2\_6 {[] <> !((SW\_L==Green) && SW\_S)}

/\* Synchronization channels \*/

chan NS\_WN = [0] of {bool};

chan NE\_EW = [0] of {bool};

chan ES\_SW = [0] of {bool};

chan NS\_SW = [0] of {bool};

chan NS\_EW = [0] of {bool};

chan NE\_WN = [0] of {bool};

chan NE\_ES = [0] of {bool};

chan EW\_WN = [0] of {bool};

chan SW\_WN = [0] of {bool};

init

{

atomic{

NS\_WN ! true;

NE\_EW ! true;

ES\_SW ! true;

NS\_SW ! true;

NS\_EW ! true;

NE\_WN ! true;

NE\_ES ! true;

SW\_WN ! true;

EW\_WN ! true;

};

}

/\* Traffic generation process \*/

active proctype gen\_t ()

{

do

:: true ->

NS\_S = !NS\_S;

:: true ->

WN\_S = !WN\_S;

:: true ->

NE\_S = !NE\_S;

:: true ->

EW\_S = !EW\_S;

:: true ->

ES\_S = !ES\_S;

:: true ->

SW\_S = !SW\_S;

od;

}

/\* NS controller \*/

active proctype NS ()

{

do

/\* Wait for resources \*/

:: if

:: NS\_S ->

atomic{NS\_WN ? true; NS\_SW ? true; NS\_EW ? true;}

NS\_L = Green;

if

/\* Wait for end of car queue \*/

:: !NS\_S -> skip;

fi;

NS\_L = Red;

atomic{NS\_WN ! true; NS\_SW ! true; NS\_EW ! true;}

:: else -> skip;

fi;

/\* Send WN that NS direction is free \*/

od;

}

/\* WN controller \*/

active proctype WN ()

{

do

:: if

:: WN\_S ->

/\* Wait for resources \*/

atomic{NS\_WN ? true; SW\_WN ? true; EW\_WN ? true; NE\_WN ? true;}

WN\_L = Green;

if

/\* Wait for end of car queue \*/

:: !WN\_S -> skip;

fi;

WN\_L = Red;

/\* Send WN that NS direction is free \*/

atomic{NS\_WN ! true; SW\_WN ! true; EW\_WN ! true; NE\_WN ! true;}

:: else -> skip;

fi;

od;

}

/\* NE controller \*/

active proctype NE ()

{

do

/\* Wait for resources \*/

:: if

:: NE\_S ->

atomic{NE\_EW ? true; NE\_WN ? true; NE\_ES ? true; }

NE\_L = Green;

if

/\* Wait for end of car queue \*/

:: !NE\_S -> skip;

fi;

NE\_L = Red;

atomic{NE\_EW ! true; NE\_WN ! true; NE\_ES ! true; }

:: else -> skip;

fi;

/\* Send WN that NS direction is free \*/

od;

}

/\* NE controller \*/

active proctype EW ()

{

do

/\* Wait for resources \*/

:: if

:: EW\_S ->

atomic{NE\_EW ? true; NS\_EW ? true; EW\_WN ? true; }

EW\_L = Green;

if

/\* Wait for end of car queue \*/

:: !EW\_S -> skip;

fi;

EW\_L = Red;

atomic{NE\_EW ! true; NS\_EW ! true; EW\_WN ! true; }

:: else -> skip;

fi;

/\* Send WN that NS direction is free \*/

od;

}

/\* ES controller \*/

active proctype ES ()

{

do

/\* Wait for resources \*/

:: if

:: ES\_S ->

atomic{ES\_SW ? true; NE\_ES ? true;}

ES\_L = Green;

if

/\* Wait for end of car queue \*/

:: !ES\_S -> skip;

fi;

ES\_L = Red;

atomic{ES\_SW ! true; NE\_ES ! true;}

:: else -> skip;

fi;

/\* Send WN that NS direction is free \*/

od;

}

/\* SW controller \*/

active proctype SW ()

{

do

/\* Wait for resources \*/

:: if

:: SW\_S ->

atomic{ES\_SW ? true; SW\_WN ? true; NS\_SW ? true;}

SW\_L = Green;

if

/\* Wait for end of car queue \*/

:: !SW\_S -> skip;

fi;

SW\_L = Red;

atomic{ES\_SW ! true; SW\_WN ! true; NS\_SW ! true;}

:: else -> skip;

fi;

/\* Send WN that NS direction is free \*/

od;

}

## LTL правила

Данные правила на языке темпоральной логики оперируют двумя базовыми понятиями:

1. G ([] – в системе Spin) – описывает свойство, которое должно выполняться всегда
2. F (<> - в системе Spin) – описывает свойство, которое должно выполнится когда-то в будущем

Правила:

Безопасность

/\*Safety\*/

ltl p {[] !((NS\_L==Green && WN\_L==Green && SW\_L==Green && EW\_L==Green)

||(WN\_L==Green && NS\_L==Green && NE\_L==Green && SW\_L==Green && EW\_L==Green)

||(EW\_L==Green && NE\_L==Green && WN\_L==Green && NS\_L==Green)

||(NE\_L==Green && WN\_L==Green && ES\_L==Green && EW\_L==Green)

||(SW\_L==Green && ES\_L==Green && WN\_L==Green && NS\_L==Green)

||(ES\_L==Green && SW\_L==Green && NE\_L==Green))}

Живость

/\*Liveness\*/

ltl p1\_1 {[] ((NS\_S && (NS\_L==Red)) -> <> (NS\_L==Green))}

ltl p1\_2 {[] ((WN\_S && (WN\_L==Red)) -> <> (WN\_L==Green))}

ltl p1\_3 {[] ((NE\_S && (NE\_L==Red)) -> <> (NE\_L==Green))}

ltl p1\_4 {[] ((EW\_S && (EW\_L==Red)) -> <> (EW\_L==Green))}

ltl p1\_5 {[] ((ES\_S && (ES\_L==Red)) -> <> (ES\_L==Green))}

ltl p1\_6 {[] ((SW\_S && (SW\_L==Red)) -> <> (SW\_L==Green))}

Справедливость

/\*Fairness\*/

ltl p2\_1 {[] <> !((NS\_L==Green) && NS\_S)}

ltl p2\_2 {[] <> !((WN\_L==Green) && WN\_S)}

ltl p2\_3 {[] <> !((NE\_L==Green) && NE\_S)}

ltl p2\_4 {[] <> !((EW\_L==Green) && EW\_S)}

ltl p2\_5 {[] <> !((ES\_L==Green) && ES\_S)}

ltl p2\_6 {[] <> !((SW\_L==Green) && SW\_S)}

# Выводы