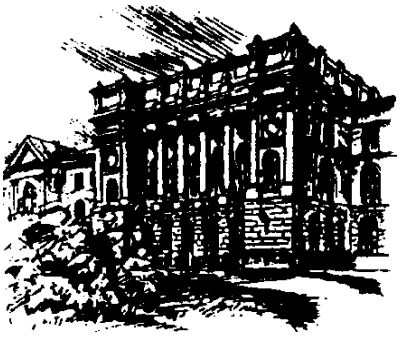
Минобрнауки России

Санкт-Петербургский государственный политехнический университет

Институт информационных технологий и управления

Кафедра «Информационные и управляющие системы»



К У Р С О В О Й П Р О Е К Т

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