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

К У Р С О В А Я Р А Б О Т А

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.

«__» _____ 2013 г.

Санкт-Петербург
2013

Оглавление

Введение

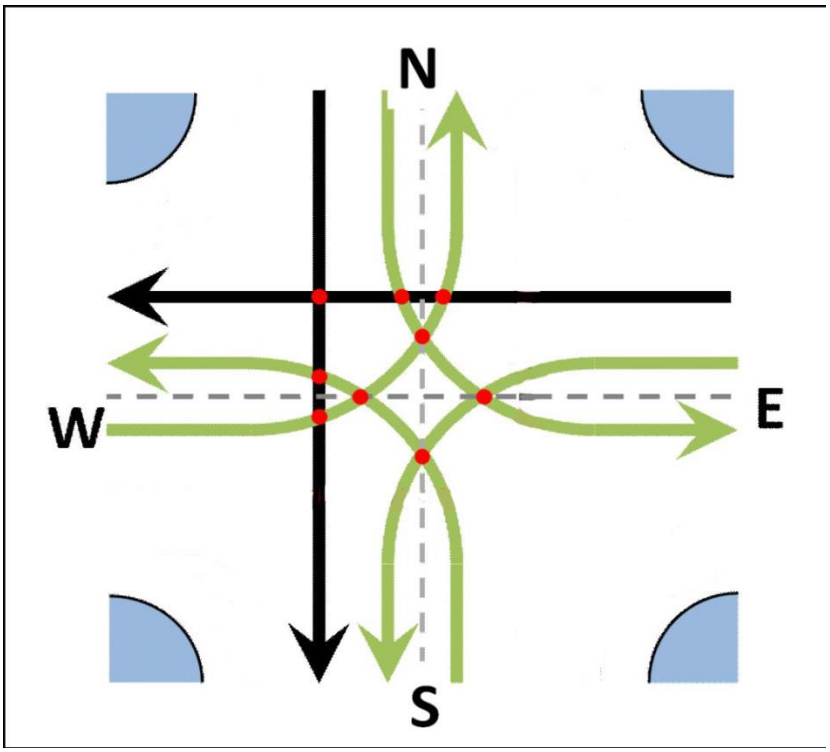
Верификация – это проверка модели (алгоритма, системы) на корректность работы, с учетом заданных правил. В настоящее время вопрос корректности работы программных продуктов является крайне актуальным. На программное обеспечивание возлагаются все новые и новые функции, при это в некоторых областях возникновение 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 варианта необходимо создать набор свойств (LTL), которые определяют корректность работы модели (например отсутствие варианта, когда движение разрешено всем), а так же описать саму модель на языке Promela.

Реализация

Основная идея

Основной идеей данной реализации является отождествление пересечений направлений движения с ресурсами, которые необходимо получить контролеру для разрешения безопасного проезда. Данных подход обладает целым рядом преимуществ, из которых можно выделить:

1) Контролеры, управляющие пересекающимися направлениями, не могут дать зеленый свет одновременно
2) «Дружественные» контроллеры имеют возможность пропускать поток независимо
3) Данный подход довольно просто в реализации

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;
/* Car crossing checkers */
bool NS_C = true;
bool WN_C = true;
bool NE_C = true;
bool EW_C = true;
bool ES_C = true;
bool SW_C = true;
/*Safety*/
ltl p0_1 {[] !(NS_L==Green && WN_L==Green && SW_L==Green && EW_L==Green)}
ltl p0_2 {[] !(WN_L==Green && NS_L==Green && NE_L==Green && SW_L==Green && EW_L==Green)}
ltl p0_3 {[] !(EW_L==Green && NE_L==Green && WN_L==Green && NS_L==Green)}
ltl p0_4 {[] !(NE_L==Green && WN_L==Green && ES_L==Green && EW_L==Green)}
ltl p0_5 {[] !(SW_L==Green && ES_L==Green && WN_L==Green && NS_L==Green)}
ltl p0_6 {[] !(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_EW = [0] of {bool};
chan NS_WN_SW = [0] of {bool};
chan NE_WN_EW = [0] of {bool};
chan NE_ES = [0] of {bool};
chan ES_SW = [0] of {bool};
init
{
        atomic{
```

```
                NS_WN_EW ! true;
                NS_WN_SW ! true;
                NE_WN_EW ! true;
                NE_ES ! true;
                ES_SW ! true;
  };
}
/* Traffic generation process */
active proctype gen_t ()
{
        do
                :: true ->
                        if
                                :: NS_C -> NS_S = !NS_S; NS_C = false;
                                :: else -> skip;
                        fi
          :: true ->
                if
                        :: WN_C -> WN_S = !WN_S; WN_C = false;
                        :: else -> skip;
                fi
                :: true ->
                if
                        :: NE_C -> NE_S = !NE_S; NE_C = false;
                        :: else -> skip;
                fi
                :: true ->
                if
                                :: EW_C -> EW_S = !EW_S; EW_C = false;
                        :: else -> skip;
                fi
                :: true ->
                if
                                :: ES_C -> ES_S = !ES_S; ES_C = false;
                        :: else -> skip;
                fi
                :: true ->
                if
```

```
                                        :: SW_C -> SW_S = !SW_S; SW_C=false;
                                :: else -> skip;
                        fi
  od;
}
/* NS controller */
active proctype NS ()
{
        do
        /* Wait for resources */
                :: if
                        :: NS_S ->
                                        NS_WN_EW ? true; NS_WN_SW ? true;
                                NS_L = Green;
                                NS_C = true;
                                if
                                        /* Wait for end of car queue */
                                        :: !NS_S -> skip;
                                fi;
                                NS_L = Red;
                                NS_WN_EW ! true; NS_WN_SW ! true;
                        :: else -> skip;
                fi;
  od;
}
/* WN controller */
active proctype WN ()
{
        do
                :: if
                        :: WN_S ->
                                /* Wait for resources */
                                NS_WN_EW ? true; NS_WN_SW ? true; NE_WN_EW ? true;
                                WN_L = Green;
                                WN_C = true;
                                if
                                /* Wait for end of car queue */
                                        :: !WN_S -> skip;
```

```
                                        fi;
                                        WN_L = Red;
                                        NS_WN_EW ! true; NS_WN_SW ! true; NE_WN_EW ! true;
                                :: else -> skip;
                        fi;
                od;
}
/* NE controller */
active proctype NE ()
{
                do
                /* Wait for resources */
                        :: if
                                :: NE_S ->
                                                NE_WN_EW  ? true; NE_ES ? true;
                                        NE_L = Green;
                                        NE_C = true;
                                        if
                                                /* Wait for end of car queue */
                                                :: !NE_S -> skip;
                                        fi;
                                        NE_L = Red;
                                        NE_WN_EW ! true; NE_ES ! true;
                                :: else -> skip;
                        fi;
   od;
}
/* NE controller */
active proctype EW ()
{
                do
                /* Wait for resources */
                        :: if
                                :: EW_S ->
                                        NS_WN_EW ? true; NE_WN_EW ? true;
                                        EW_L = Green;
                                        EW_C = true;
                                        if
```

```
                                        /* Wait for end of car queue */
                                        :: !EW_S -> skip;
                                    fi;
                                    EW_L = Red;
                                    NS_WN_EW ! true; NE_WN_EW ! true;
                                :: else -> skip;
                        fi;
    od;
}
/* ES controller */
active proctype ES ()
{
        do
        /* Wait for resources */
                :: if
                        :: ES_S ->
                                ES_SW ? true; NE_ES ? true;
                                ES_L = Green;
                                ES_C = true;
                                if
                                        /* Wait for end of car queue */
                                        :: !ES_S -> skip;
                                fi;
                                ES_L = Red;
                                ES_SW ! true; NE_ES ! true;
                        :: else -> skip;
                fi;
    od;
}
/* SW controller */
active proctype SW ()
{
        do
        /* Wait for resources */
                :: if
                        :: SW_S ->
                                NS_WN_SW ? true; ES_SW ? true;
                                SW_L = Green;
```

```
                    SW_C = true;
                    if
                        /* Wait for end of car queue */
                        :: !SW_S -> skip;
                    fi;
                    SW_L = Red;
                    NS_WN_SW ! true; ES_SW ! true;
                :: else -> skip;
            fi;
    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 данной системы:

Безопасность

Для данной модели правила безопасности 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»

1) NS - `{[] !(NS_L==Green && WN_L==Green && SW_L==Green && EW_L==Green)}`
2) WN – `{ [] !(WN_L==Green && NS_L==Green && NE_L==Green && SW_L==Green && EW_L==Green)}`
3) NE - `{[] !(NE_L==Green && WN_L==Green && ES_L==Green && EW_L==Green)}`
4) EW - `{[] !(EW_L==Green && NE_L==Green && WN_L==Green && NS_L==Green)}`
5) SW - `{[] !(SW_L==Green && ES_L==Green && WN_L==Green && NS_L==Green)}`
6) ES - `{[] !(ES_L==Green && SW_L==Green && NE_L==Green)}`

Живость

Для данной модели правила живости звучат на естественном языке следующим образом: «Всегда выполняется, если светофор горит красным и датчик показывает наличие машин, следовательно в будущем данный светофор будет гореть зеленым»

1) NS - `{[] ((NS_S && (NS_L==Red)) -> <> (NS_L==Green))}`
2) WN - `{[] ((WN_S && (WN_L==Red)) -> <> (WN_L==Green))}`
3) NE - `{[] ((NE_S && (NE_L==Red)) -> <> (NE_L==Green))}`
4) EW - `{[] ((EW_S && (EW_L==Red)) -> <> (EW_L==Green))}`
5) SW - `{[] ((SW_S && (SW_L==Red)) -> <> (SW_L==Green))}`
6) ES - `{[] ((ES_S && (ES_L==Red)) -> <> (ES_L==Green))}`

Справедливость

Для данной модели правила справедливости звучат на естественном языке следующим образом: «Невозможна такая ситуация, что в конкретном направлении движется непрерывный поток машин(т.е. светофор должен неопределенно часто менять свой сигнал на запрещающий)»

1) NS - `{[]<> !((NS_L==Green) && NS_S)}`
2) WN - `{[]<> !((WN_L==Green) && WN_S)}`
3) EW - `{[]<> !((NE_L==Green) && NE_S)}`
4) NE - `{[]<> !((EW_L==Green) && EW_S)}`
5) SW - `{[]<> !((ES_L==Green) && ES_S)}`
6) ES - `{[]<> !((SW_L==Green) && SW_S)}`

Результаты моделирования

Моделирование показало, что все проверки на безопасность проходят. А для каждой из проверок на живучесть и справедливость находится по одной трассе – опровержению. Эта трасса соответствует случаю, когда случайный выбор направления в генераторе трафика всегда выбирает одно и то же направление. Однако понятно, что с точки зрения законов естественного мира, данная ситуация никогда не может быть воспроизведена. Далее приводится вывод системы Spin:

```
spin -a  var1.pml

ltl p0_1: [] (! (((((NS_L==Green)) && ((WN_L==Green))) && ((SW_L==Green))) && ((EW_L==Green))))

ltl p0_2: [] (! ((((((WN_L==Green)) && ((NS_L==Green))) && ((NE_L==Green))) && ((SW_L==Green))) &&
((EW_L==Green))))

ltl p0_3: [] (! (((((EW_L==Green)) && ((NE_L==Green))) && ((WN_L==Green))) && ((NS_L==Green))))

ltl p0_4: [] (! (((((NE_L==Green)) && ((WN_L==Green))) && ((ES_L==Green))) && ((EW_L==Green))))
```

```
ltl p0_5: [] (! (((((SW_L==Green)) && ((ES_L==Green))) && ((WN_L==Green))) && ((NS_L==Green))))

ltl p0_6: [] (! ((((ES_L==Green)) && ((SW_L==Green))) && ((NE_L==Green))))

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))))

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))))

  the model contains 18 never claims: p2_6, p2_5, p2_4, p2_3, p2_2, p2_1, p1_6, p1_5, p1_4, p1_3, p1_2, p1_1,
p0_6, p0_5, p0_4, p0_3, p0_2, p0_1

  only one claim is used in a verification run

  choose which one with ./pan -a -N name (defaults to -N p0_1)

gcc -DMEMLIM=1024 -O2 -DXUSAFE -w -o pan pan.c

./pan -m10000  -a -N p0_1

Pid: 4242

Depth=    3586 States=    1e+06 Transitions= 3.89e+06 Memory=   212.616       t=      1.6 R=   6e+05

Depth=    3586 States=    2e+06 Transitions= 7.62e+06 Memory=   296.601       t=     3.17 R=   6e+05

(Spin Version 6.2.5 -- 3 May 2013)

            + Partial Order Reduction

Full statespace search for:

            never claim         + (p0_1)

            assertion violations           + (if within scope of claim)

            acceptance   cycles           + (fairness disabled)

            invalid end states - (disabled by never claim)

State-vector 148 byte, depth reached 3586, errors: 0

  2997002 states, stored

  8548319 states, matched

 11545321 transitions (= stored+matched)

        0 atomic steps

hash conflicts:    328613 (resolved)
```

```
Stats on memory usage (in Megabytes):
  503.037	equivalent memory usage for states (stored*(State-vector + overhead))
  251.951	actual memory usage for states (compression: 50.09%)
		state-vector as stored = 60 byte + 28 byte overhead
  128.000	memory used for hash table (-w24)
    0.534	memory used for DFS stack (-m10000)
  380.292	total actual memory usage
unreached in init
		(0 of 7 states)
unreached in proctype gen_t
		var1.pml:127, state 52, "-end-"
		(1 of 52 states)
unreached in proctype NS
		var1.pml:148, state 20, "-end-"
		(1 of 20 states)
unreached in proctype WN
		var1.pml:169, state 22, "-end-"
		(1 of 22 states)
unreached in proctype NE
		var1.pml:190, state 20, "-end-"
		(1 of 20 states)
unreached in proctype EW
		var1.pml:211, state 20, "-end-"
		(1 of 20 states)
unreached in proctype ES
		var1.pml:232, state 20, "-end-"
		(1 of 20 states)
unreached in proctype SW
		var1.pml:253, state 20, "-end-"
		(1 of 20 states)
unreached in claim p0_1
		_spin_nvr.tmp:8, state 10, "-end-"
		(1 of 10 states)
pan: elapsed time 4.8 seconds
No errors found -- did you verify all claims?
```

Выводы

В рамках данной курсовой работы были исследованы подходы к верификации распределенных программ. Для модельной задачи были определены LTL формулы, описывающие корректное поведение системы. Далее модель была описана на языке Promela в системе Spin. После чего была проведена верификация с учетом описанных в ответе LTL формул. По результатам верификации можно судить о корректности модели.

Таким образом можно говорить об успешном выполнении задания и освоении инструментов верификации. Данные знания являются актуальными и могут применяться в большом количестве практик современного программирования.