Минобрнауки России

Санкт-Петербургский Государственный Политехнический Университет

Институт Информационных Технологий и Управления

Кафедра «Информационные и Управляющие Системы»

КУРСОВАЯ РАБОТА

Разработка контроллера светофоров и его верификация

по дисциплине «Распределенные алгоритмы и протоколы»

Выполнил

студент гр. 63504/12 Соловьев И.С.

Руководитель

ст. преподаватель Шошмина И.В.

Санкт-Петербург

2013

Оглавление

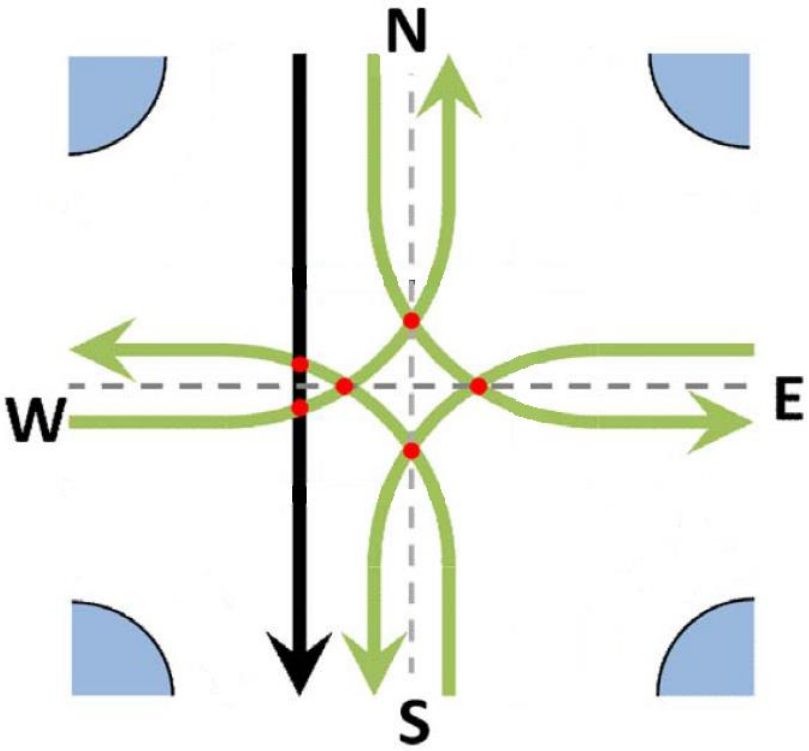
Введение

В данной курсовой работе строится модель контроллера светофоров для управления движением на перекрестке. При этом каждое направление на перекрестке контролируется своим светофором. На каждой полосе движения установлены датчики, которые фиксируют появление автомобиля на соответствующем направлении. В отсутствие автомобилей светофор горит красным, а при появлении автомобиля процесс, контролирующий данное направление, начинает борьбу за ресурсы с тем, чтобы позволить автомобилям проехать.

Постановка задачи

Вариант (2, 5, 13)

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



Модель системы строится на языке Promela. Для верификации используется системы SPIN и графический интерфейс iSpin.

Реализация

Основная идея реализации модели

1. Разделяемые ресурсы данной модели – пересечения направлений движения (отмечены красными точками на рисунке). Такой подход позволяет исключить одновременный проезд автомобилей по пересекающимся направлениям, но, в то же время, позволяет одновременное движение по независимым направлениям.
2. Траффик генерируется дополнительным процессом (trafficManager), который имитирует появление машин на разных направлениях случайным образом.
3. Взаимодействие между процессами происходит за счет сигналов.
4. Алгоритм проезда автомобиля через перекресток:
    a. Процесс trafficManager генерирует сигнал о том, что на определенном направлении (например, WN) появился автомобиль.
    b. Процесс dirWN, отвечающий за заданное направление, принимает сигнал и начинает борьбу за ресурсы.
    c. Когда все необходимые ресурсы захвачены, процесс dirWN сигнализирует о том, что машины могут проезжать.
    d. Получив этот сигнал, процесс trafficManager в случайный момент времени имитирует ситуацию, при которой все автомобили в данном направлении проехали через перекресток. Сигнал об этом поступает обратно процессу dirWN.
    e. Получив последний сигнал, процесс dirWN освобождает ресурсы и снова ожидает появления автомобилей на заданном направлении.

Модель на языке Promela

```
/*
   Task numbers:
   2 - WN, NE;
   5 - NS, SW;
   13 - NE, ES.
*/

/* Traffic lights states */
mtype = {RED, GREEN};

/* Directions mtype */
mtype = {WN, NE, NS, SW, ES};

/* Channels */
chan channel_WN_NS_SW = [1] of {bit};
chan channel_WN_NE = [1] of {bit};
chan channel_NE_ES = [1] of {bit};
chan channel_ES_SW = [1] of {bit};

/* Channel for sensors' states */
chan sensorChannel = [5] of {mtype, bit};

/* Channels for car passes */
chan carsPassWN = [1] of {bit};
```

```
chan carsPassNE = [1] of {bit};
chan carsPassNS = [1] of {bit};
chan carsPassSW = [1] of {bit};
chan carsPassES = [1] of {bit};

/* Traffic manager process */
proctype trafficManager () {
   bool sensorWN = false;
   bool sensorNE = false;
   bool sensorNS = false;
   bool sensorSW = false;
   bool sensorES = false;

end:
   do
      :: (sensorWN && (len (carsPassWN) != 0))->
         carsPassWN ? 1;
wnFalse:
         sensorWN = false;
         sensorChannel ! WN, 0;
      :: (sensorNE && (len (carsPassNE) != 0)) ->
         carsPassNE ? 1;
neFalse:
         sensorNE = false;
         sensorChannel ! NE, 0;
      :: (sensorNS && (len (carsPassNS) != 0)) ->
         carsPassNS ? 1;
nsFalse:
         sensorNS = false;
         sensorChannel ! NS, 0;
      :: (sensorSW && (len (carsPassSW) != 0)) ->
         carsPassSW ? 1;
swFalse:
         sensorSW = false;
         sensorChannel ! SW, 0;
      :: (sensorES && (len (carsPassES) != 0)) ->
         carsPassES ? 1;
esFalse:
         sensorES = false;
         sensorChannel ! ES, 0;
      :: !sensorWN ->
wnTrue:
         sensorWN = true;
         sensorChannel ! WN, 1;
      :: !sensorNE ->
neTrue:
         sensorNE = true;
         sensorChannel ! NE, 1;
      :: !sensorNS ->
nsTrue:
         sensorNS = true;
         sensorChannel ! NS, 1;
      :: !sensorSW ->
swTrue:
         sensorSW = true;
         sensorChannel ! SW, 1;
      :: !sensorES ->
esTrue:
         sensorES = true;
```

```
        sensorChannel ! ES, 1;
    od;
}

/* WN direction process */
proctype dirWN () {
    mtype lightWN = RED;
end:
    do
        ::
        /* Wait for the resources */
        sensorChannel ? WN, 1;
        channel_WN_NS_SW ? 1; channel_WN_NE ? 1;
        /* We can go! */
green:
        lightWN = GREEN;
        carsPassWN ! 1;
        sensorChannel ? WN, 0;
red:
        lightWN = RED;
        /* Return resources */
        channel_WN_NS_SW ! 1; channel_WN_NE ! 1;
    od;
}

/* NE direction process */
proctype dirNE () {
    mtype lightNE = RED;
end:
    do
        ::
        /* Wait for the resources */
        sensorChannel ? NE, 1;
        channel_NE_ES ? 1; channel_WN_NE ? 1;
        /* We can go! */
green:
        lightNE = GREEN;
        carsPassNE ! 1;
        sensorChannel ? NE, 0;
red:
        lightNE = RED;
        /* Return resources */
        channel_NE_ES ! 1; channel_WN_NE ! 1;
    od;
}

/* NS direction process */
proctype dirNS () {
    mtype lightNS = RED;
end:
    do
        ::
        /* Wait for the resources */
        sensorChannel ? NS, 1;
        channel_WN_NS_SW ? 1;
        /* We can go! */
green:
        lightNS = GREEN;
        carsPassNS ! 1;
```

```
      sensorChannel ? NS, 0;
red:
      lightNS = RED;
      /* Return resources */
      channel_WN_NS_SW ! 1;
   od;
}

/* SW direction process */
proctype dirSW () {
   mtype lightSW = RED;
end:
   do
      ::
      /* Wait for the resources */
      sensorChannel ? SW, 1;
      channel_ES_SW ? 1; channel_WN_NS_SW ? 1;
      /* We can go! */
green:
      lightSW = GREEN;
      carsPassSW ! 1;
      sensorChannel ? SW, 0;
red:
      lightSW = RED;
      /* Return resources */
      channel_ES_SW ! 1; channel_WN_NS_SW ! 1;
   od;
}

/* ES direction process */
proctype dirES () {
   mtype lightES = RED;
end:
   do
      ::
      /* Wait for the resources */
      sensorChannel ? ES, 1;
      channel_ES_SW ? 1; channel_NE_ES ? 1;
      /* We can go! */
green:
      lightES = GREEN;
      carsPassES ! 1;
      sensorChannel ? ES, 0;
red:
      lightES = RED;
      /* Return resources */
      channel_ES_SW ! 1; channel_NE_ES ! 1;
   od;
}

/* Init process */
init {
   /* All resources are freed at start */
   atomic {
      channel_WN_NS_SW ! 1;
      channel_WN_NE ! 1;
      channel_NE_ES ! 1;
      channel_ES_SW ! 1;
```

```
        run trafficManager ();

        run dirWN ();
        run dirNE ();
        run dirNS ();
        run dirSW ();
        run dirES ();
    }
}
```

LTL-формулы

Safety

Для данной модели свойства safety на естественном языке звучат следующим образом: «Не должно возникнуть ситуации, при которой зеленый свет зажегся одновременно на двух или более пересекающихся направлениях».

LTL-формула в системе Spin:

```
ltl s {[] !((dirNS@green && dirSW@green && dirWN@green)
                || (dirWN@green && dirSW@green && dirNS@green && dirNE@green)
                || (dirNE@green && dirES@green && dirWN@green)
                || (dirES@green && dirSW@green && dirNE@green)
                || (dirSW@green && dirES@green && dirWN@green && dirNS@green))};
```

Fairness & Liveness

Доказывать свойства fairness и liveness отдельно для данной модели не представляется возможным из-за того, что система Spin обладает «случайным» планировщиком процессов, который не гарантирует выполнения всех процессов. Таким образом, может сложиться ситуация, при которой один или несколько процессов будут выполняться всегда, в то время как все остальные процессы будут простаивать. Ясно, что в такой ситуации свойства fairness и liveness для простаивающих процессов выполняться не будут.

Таким образом, свойства fairness и liveness надо доказывать вместе. На естественном языке данные свойства звучат следующим образом: «При появлении автомобиля на каком-либо направлении ему обязательно предоставится возможность проехать (возможно, через какое-то время), при ограничении, что в каждом направлении не движется непрерывный поток автомобилей».

LTL-формулы для каждого направления имеют следующий вид:

1. `ltl wn { ([] <> !(dirWN@green && trafficManager@wnTrue)) -> ([] <> (!(dirWN@green && trafficManager@wnTrue) && ([] ((trafficManager@wnTrue && dirWN@red) -> <> dirWN@green)))) };`
2. `ltl ne { ([] <> !(dirNE@green && trafficManager@neTrue)) -> ([] <> (!(dirNE@green && trafficManager@neTrue) && ([] ((trafficManager@neTrue && dirNE@red) -> <> dirNE@green)))) };`
3. `ltl ns { ([] <> !(dirNS@green && trafficManager@nsTrue)) -> ([] <> (!(dirNS@green && trafficManager@nsTrue) && ([] ((trafficManager@nsTrue && dirNS@red) -> <> dirNS@green)))) };`

4. ltl sw { ([] <> !(dirSW@green && trafficManager@swTrue)) -> ([] <> (!(dirSW@green && trafficManager@swTrue) && ([] ((trafficManager@swTrue && dirSW@red) -> <> dirSW@green)))) };
5. ltl es { ([] <> !(dirES@green && trafficManager@esTrue)) -> ([] <> (!(dirES@green && trafficManager@esTrue) && ([] ((trafficManager@esTrue && dirES@red) -> <> dirES@green)))) };

Результаты моделирования

Моделирование показало, что проверка свойств safety и fairness & liveness прошла успешно.

Далее приводится вывод системы Spin при проверке свойства ns:

```
gcc -DMEMLIM=2048 -O2 -DXUSAFE -w -o pan pan.c
./pan -m200000  -a -N ns
Pid: 3025
Depth=   50858 States=    1e+06 Transitions= 3.01e+06 Memory=   132.878      t=    2.51 R=   4e+05
Depth=   53196 States=    2e+06 Transitions= 6.44e+06 Memory=   203.484      t=    5.26 R=   4e+05
Depth=   53196 States=    3e+06 Transitions= 9.82e+06 Memory=   272.234      t=       8 R=   4e+05
Depth=   53196 States=    4e+06 Transitions= 1.31e+07 Memory=   338.640      t=    10.7 R=   4e+05
Depth=   53196 States=    5e+06 Transitions= 1.59e+07 Memory=   425.847      t=      13 R=   4e+05
Depth=   53196 States=    6e+06 Transitions= 1.91e+07 Memory=   495.476      t=    15.7 R=   4e+05
Depth=   53196 States=    7e+06 Transitions= 2.24e+07 Memory=   564.812      t=    18.8 R=   4e+05
Depth=   53196 States=    8e+06 Transitions= 2.53e+07 Memory=   641.863      t=    21.4 R=   4e+05
Depth=   53196 States=    9e+06 Transitions= 2.86e+07 Memory=   711.296      t=    24.4 R=   4e+05
Depth=   53196 States=    1e+07 Transitions= 3.18e+07 Memory=   777.898      t=    27.4 R=   4e+05
Depth=   53196 States=  1.1e+07 Transitions= 3.52e+07 Memory=   845.769      t=    30.3 R=   4e+05
Depth=   53196 States=  1.2e+07 Transitions= 3.82e+07 Memory=   923.503      t=    33.2 R=   4e+05
Depth=  107314 States=  1.3e+07 Transitions=  4.3e+07 Memory=  1008.269      t=    37.6 R=   3e+05
Depth=  114366 States=  1.4e+07 Transitions= 5.02e+07 Memory=  1109.343      t=    44.1 R=   3e+05
Depth=  114366 States=  1.5e+07 Transitions= 5.77e+07 Memory=  1210.027      t=    50.7 R=   3e+05
Depth=  114366 States=  1.6e+07 Transitions= 6.45e+07 Memory=  1312.663      t=    57.2 R=   3e+05
Depth=  114366 States=  1.7e+07 Transitions= 7.18e+07 Memory=  1414.617      t=    63.7 R=   3e+05
Depth=  114366 States=  1.8e+07 Transitions= 7.89e+07 Memory=  1515.788      t=    70.1 R=   3e+05
Depth=  114366 States=  1.9e+07 Transitions= 8.59e+07 Memory=  1617.839      t=    77.3 R=   2e+05
Depth=  114366 States=    2e+07 Transitions= 9.31e+07 Memory=  1719.109      t=    84.2 R=   2e+05
Depth=  114366 States=  2.1e+07 Transitions=    1e+08 Memory=  1820.183      t=    90.9 R=   2e+05

(Spin Version 6.2.5 -- 3 May 2013)
        + Partial Order Reduction

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

State-vector 116 byte, depth reached 114366, errors: 0
 17871651 states, stored (2.19047e+07 visited)
 84460462 states, matched
1.063652e+08 transitions (= visited+matched)
     27 atomic steps
hash conflicts:  25135712 (resolved)
```

Stats on memory usage (in Megabytes):
2249.773 equivalent memory usage for states (stored(State-vector + overhead))*
1841.695 actual memory usage for states (compression: 81.86%)
state-vector as stored = 92 byte + 16 byte overhead
64.000 memory used for hash table (-w24)
6.867 memory used for DFS stack (-m200000)
1912.175 total actual memory usage
pan: elapsed time 97.4 seconds
No errors found -- did you verify all claims?

Выводы

В рамках данной курсовой работы были исследованы подходы к верификации распределенных программ. Заданная модель была описана на языке Promela в системе Spin. Для модели были определены LTL формулы, описывающие корректное поведение системы, после чего была проведена верификация. Верификация всех LTL-формул прошла успешно, что подтверждает корректность модели.

Список использованных источников

Ю.Г. Карпов, И.В. Шошмина Верификация распределенных систем – СПб.: Издательство Политехнического университета, 2011.

Thomas Wahl – Fairness and Liveness
URL: http://www.ccs.neu.edu/home/wahl/Publications/fairness.pdf (дата обращения: 06.12.2013).