Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Высшая школа прикладной математики и вычислительной физики

Отчет по дисциплине «Верификация распределенных алгоритмов и систем» по курсовому проекту «Разработка контроллера светофоров и его верификация»

Выполнил: Студент гр. 3640102/00201
Чепулис М.А.

Преподаватель:
Шошмина И.В.

Санкт-Петербург
2020

Оглавление

Введение

Постановка задачи

Рассматривается сложный перекрёсток с четырьмя двусторонними направлениями движения (рис. 1). В каждом направлении имеются три полосы движения для трёх траекторий движения: для проезда прямо по перекрёстку, для поворота направо, для поворота налево. Движение по каждой траектории регулируется своим светофором: один для проезда прямо через перекрёсток, один (стрелка) – для поворота направо, один (стрелка) – для поворота налево.

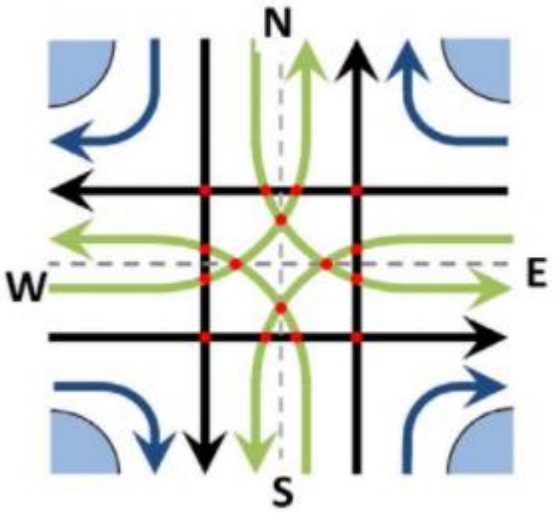

Рисунок 1 Схема сложно перекрёстка

Светофор, регулирующий движение по любой траектории, загорается зелёным, если, во-первых, есть запрос от машин в соответствующей полосе и, во-вторых, по всем тем направлениям, с которыми существует пересечении, движение запрещено – на них горит красный сигнал.

Необходимо разработать и верифицировать средствами Spin [2] модель контроллера светофоров для управления переездом через перекрёсток. Светофором для каждого конкретного направления должен регулировать отдельный процесс.

Индивидуальный вариант

В моём варианте задания присутствуют только направления North – South (NS), East – South (ES), South – West (SW) и West – North (WN). Тогда перекрёсток представляется схемой на Рисунке 2.

Пронумеруем точки возможных пересечений в следующем порядке:

ES-SW (1), NS-WN (2), SW-WN (3), NS-SW (4).

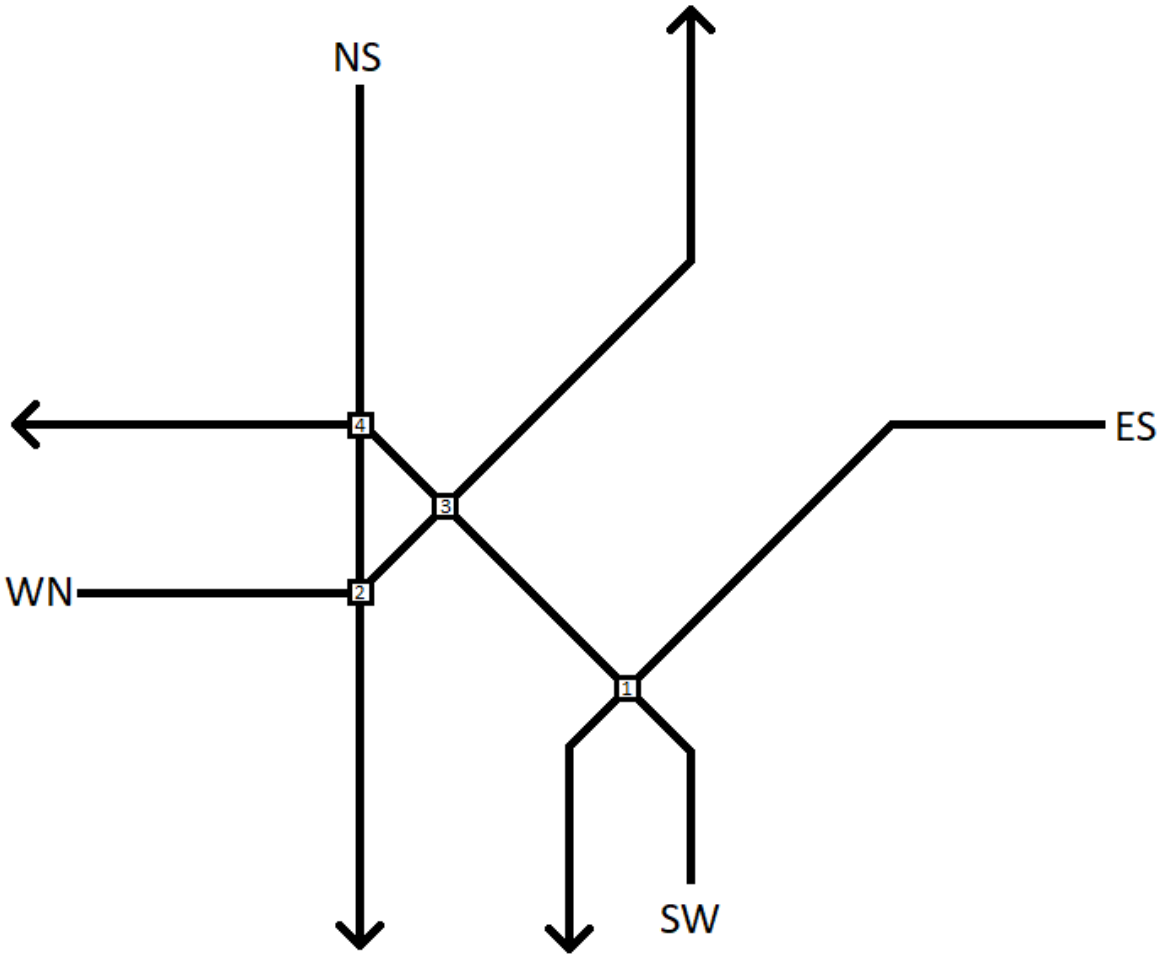

Рисунок 2 Схема индивидуального задания

Построение модели

Построение модели будет производиться на языке Promela [4].

Полный код модели можно посмотреть в Приложении 1

Переменные `X_req` отвечают за запрос проезда по направлению X.

Переменные `X_light` отвечают за цвет светофора на направлении X.

Переменные `X_Y_lock` отвечают за блокировку пересечения направлений X и Y (семантика блокировки mutex)

Логически модель можно разделить на 3 блока:

- Внешняя среда,
- Контроллеры светофоров,
- Контроллер справедливого взаимодействия.

Процесс внешней среды отвечает за приезжающие на перекрёсток машины.

Контроллер светофоров должен корректно захватить все необходимые критические зоны, а после их захвата пропустить автомобиль и освободить захваченные ресурсы.

Контроллер справедливого взаимодействия следит, чтобы ни один из процессов не «голодал».

Рассмотрим каждый блок подробнее.

Процесс внешней среды

Процесс внешней среды состоит из четырёх независимых процессов-датчиков. Каждый из которых фиксирует приезд машины на своём направлении.

Когда на направление (*X*) движения приезжает машина датчик данного направления устанавливает флаг `X_req`, после чего становится неактивным до тех пор, пока запрос не обработают, и вновь не активируют датчик.

Пример реализации процесса внешней среды:

```
active proctype NS_detector(){
do
  :: NS_req = true;
     NS_detector_can_work = false;
     (NS_detector_can_work);
od;
}
```

Контроллеры светофоров

В задачи контроллера светофоров входит управление светом светофора (красный/зелёный) и захват и освобождение критических зон.

Когда все необходимые пересечения захвачены светофор включает зелёный свет, тем самым пропуская машину. После чего переключается снова на красный, обнуляет запрос проезда, освобождает критические ресурсы и переходит к ожиданию нового запроса.

Ниже представлен пример реализации контролера светофора для направления NS

```
active proctype NS_lighter_controller(){
do
  :: (NS_req) ->
     NS_WN_lock ! true;
     NS_SW_lock ! true;

     NS_light = green;
     NS_light = red;

     NS_req = false;

     NS_WN_lock ? true;
     NS_SW_lock ? true;
od;
```

```
}
```

Для избегания взаимных блокировок все контроллеры светофоров захватывают пересечения (критические ресурсы) только в порядке увеличения их номера (см. рис.2)

В Promela mutex можно представить, как канал с единичной глубиной [5].

```
chan ES_SW_lock = [1] of {bool};
```

Тогда захватом будет являться посылка сообщения, а освобождением – получение сообщения.

Контроллер справедливого взаимодействия

В его задачу входит активация датчиков, когда будут обработаны все запросы проезда.

Пример реализации контроллера взаимодействия:

```
active proctype justice_adjuster(){
do
  ::  (  !NS_detector_can_work  ||  !ES_detector_can_work  ||
!SW_detector_can_work || !WN_detector_can_work );
      ( !NS_req && !ES_req && !SW_req && !WN_req );

      NS_detector_can_work = true;
      ES_detector_can_work = true;
      SW_detector_can_work = true;
      WN_detector_can_work = true;
od;
}
```

Сначала он ожидает, пока какой-нибудь из датчиков перейдёт в режим ожидания. После того, как все запросы будут обработаны он выводит все датчики из режима ожидания.

Он не позволяет какому-нибудь датчику вновь отправлять запрос на проезд до тех пор, пока не будут обработаны все текущие запросы. Это гарантирует, что все запросы проезда будут обработаны и ни одно из направлений не будет «голодать».

Верификация свойств

Проверяются три основные свойства модели: Безопасность, Живость, Справедливость.

Данные требования можно сформулировать в формате LTL [3].

Безопасность

Это свойство означает, что никогда не будет разрешён проезд в пересекающихся направлениях [1].

Сформулируем данное свойство для направления South-West(SW).

$$\mathrm{G\,!\,(SW\_light == green\ and\ (ES_{light} == green\ or\ NS_{light} == geen\ or\ WN_{light} == green))}$$

Аналогично можно сформулировать и для остальных направлений.

Справедливость

Предполагаем, что каждый сенсор устанавливается в false неопределенно часто, т.е. в каждом направлении не движется непрерывный поток машин [1].

Сформулируем данное свойство для направления SW.

$$\mathrm{GF\,!\,(SW\_light == green\ and\ SW\_req)}$$

Аналогично можно сформулировать и для остальных направлений.

Живость

При появлении машины, ей всегда представится возможность проезда в нужном направлении (возможно, не сразу) [1].

Сформулируем данное свойство для направления SW.

$$G\ (SW\_req\ and\ SW\_light == red \rightarrow F(SW\_light\ == green))$$

Аналогично можно сформулировать и для остальных направлений.

Запись свойств в Spin

Определения

Проверяемые в Spin свойства должны содержать атомарные предикаты, потому определим их следующим образом:

```
#define ns_green (NS_light == green)
#define es_green (ES_light == green)
#define sw_green (SW_light == green)
#define wn_green (WN_light == green)
#define ns_red (NS_light == red)
#define es_red (ES_light == red)
#define sw_red (SW_light == red)
#define wn_red (WN_light == red)
#define ns_req (NS_req == true)
#define es_req (ES_req == true)
#define sw_req (SW_req == true)
#define wn_req (WN_req == true)
```

Безопасность

Тогда свойства безопасности для каждого из направлений будет представлено в следующем виде:

```
[]   ! (ns_green  && (sw_green  ||  wn_green))
[]   ! (es_green  && sw_green)
```

```
[]   ! (sw_green  && (es_green  ||  ns_green || wn_green))
[]   ! (wn_green  && (ns_green  ||  sw_green))
```

Справедливость

Ограничения справедливости для каждого из направлений примут следующий вид:

```
[]<>!(ns_green && ns_req)
[]<>!(es_green && es_req)
[]<>!(sw_green && sw_req)
[]<>!(wn_green && wn_req):
```

Живость

Свойства живости будут выражаться так:

```
[]  ((ns_req  && ns_red)  -> <>  (ns_green))
[]  ((es_req  && es_red)  -> <>  (es_green))
[]  ((sw_req  && sw_red)  -> <>  (sw_green))
[]  ((wn_req  && wn_red)  -> <>  (wn_green))
```

Заключение

В ходе данной работы была разработана и реализованная на Promella модель контроллера. При помощи пакета Spin была проведена проверка свойств безопасности, справедливости и живости. Построенная модель соответствует всем предъявляемым требованиям,

Список использованных источников

1. Карпов Ю. Г. Верификация распределенных систем: учеб. пособие / Ю. Г. Карпов, И . В. Шошмина. — СПб. : Изд-во Политехн. ун-та, 2011. — 212 с.
2. SPIN home page. // [Электронный ресурс] Режим доступа: http://spinroot.com/spin/whatispin.html (дата обращения ноябрь 2020)
3. Linear temporal logic // [Электронный ресурс] Режим доступа: https://en.wikipedia.org/wiki/Linear_temporal_logic (дата обращения ноябрь 2020)
4. Документация Promella // [Электронный ресурс] Режим доступа: http://spinroot.com/spin/Man/chan.html (дата обращения ноябрь 2020)
5. Документация Promella по chan // [Электронный ресурс] Режим доступа: https: http://spinroot.com/spin/Man/chan.html (дата обращения ноябрь 2020)

Приложение 1

Модель на языке Promela

```
mtype = {red, green};

mtype NS_light = red;
mtype ES_light = red;
mtype SW_light = red;
mtype WN_light = red;

/* mutex */
chan ES_SW_lock = [1] of {bool};
chan NS_WN_lock = [1] of {bool};
chan SW_WN_lock = [1] of {bool};
chan NS_SW_lock = [1] of {bool};

bool NS_detector_can_work = true;
bool ES_detector_can_work = true;
bool SW_detector_can_work = true;
bool WN_detector_can_work = true;

bool NS_req = false;
bool ES_req = false;
bool SW_req = false;
bool WN_req = false;


active proctype justice_adjuster(){
do
  ::   (   !NS_detector_can_work   ||   !ES_detector_can_work   ||
!SW_detector_can_work || !WN_detector_can_work );
      ( !NS_req && !ES_req && !SW_req && !WN_req );

      NS_detector_can_work = true;
      ES_detector_can_work = true;
      SW_detector_can_work = true;
      WN_detector_can_work = true;
od;
}
```

```
/* detectors */
active proctype NS_detector(){
do
  :: NS_req = true;
     NS_detector_can_work = false;
     (NS_detector_can_work);
od;
}


active proctype ES_detector(){
do
  :: ES_req = true;
     ES_detector_can_work = false;
     (ES_detector_can_work);
od;
}


active proctype SW_detector(){
do
  :: SW_req = true;
     SW_detector_can_work = false;
     (SW_detector_can_work);
od;
}


active proctype WN_detector(){
do
  :: WN_req = true;
     WN_detector_can_work = false;
     (WN_detector_can_work);
od;
}

/* controllers */
active proctype NS_lighter_controller(){
```

```
do
  :: (NS_req) ->
     NS_WN_lock ! true;
     NS_SW_lock ! true;

     NS_light = green;
     NS_req = false;
     NS_light = red;

     NS_WN_lock ? true;
     NS_SW_lock ? true;
od;
}


active proctype ES_lighter_controller(){
do
  :: (ES_req) ->
     ES_SW_lock ! true;

     ES_light = green;
     ES_req = false;
     ES_light = red;

     ES_SW_lock ? true;
od;
}


active proctype SW_lighter_controller(){
do
  :: (SW_req) ->
     ES_SW_lock ! true;
     SW_WN_lock ! true;
     NS_SW_lock ! true;

     SW_light = green;
     SW_req = false;
     SW_light = red;
```

```
        ES_SW_lock ? true;
        SW_WN_lock ? true;
        NS_SW_lock ? true;
    od;
    }


    active proctype WN_lighter_controller(){
    do
      :: (WN_req) ->
        NS_WN_lock ! true;
        NS_SW_lock ! true;

        WN_light = green;
        WN_req = false;
        WN_light = red;

        NS_WN_lock ? true;
        NS_SW_lock ? true;
    od;
    }
```