Санкт-Петербургский государственный политехнический университет

Отчет по курсовой работе

по курсу «Верификация программ»

«Разработка контроллера светофоров на перекрёстке и его верификация»

Студент:	Руцкий В. В.
Группа:	5057/2
Вариант:	9, 13
Преподаватель:	Шошмина И. В.

Санкт-Петербург 2010

Содержание

1 Постановка задачи

Задан автомобильный перекрёсток, его конфигурация показана на рис. 1 (вариант 9, 13). На

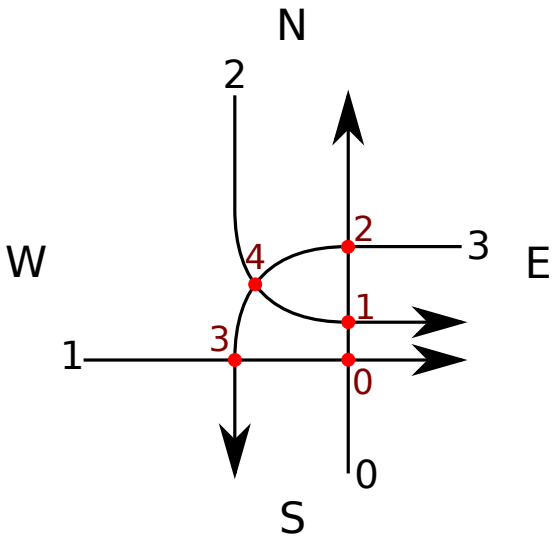

Рис. 1: Схема перекрёстка

перекрёстке возможны следующие направления движения транспорта:

- S → N,
- W → E,
- N → E,
- E → S.

Каждое направление движения регулируется собственным светофором. На каждом направлении движения установлены датчики, фиксирующие наличие автомобилей.

При отсутствии автомобилей на полосе, соответствующий светофор должен гореть красным светом. При появлении машин на полосе, соответствующий светофор должен убедиться, что по всем полосам, пересекающим текущую, движение транспорта запрещено (их светофоры горят красным), и загореться зелёным светом, пропуская машины. При наличии машин на различных полосах, светофоры должны периодически переключаться, обеспечивая равномерный пропуск машин по всем направлениям.

Требуется разработать модель управления данным светофором на языке *Promela*, удовлетворящую указанным выше требованиям, и верифицировать её корректность с помощью пакета *Spin*[1].

2 Решение

2.1 Моделирование внешней среды

Внешняя среда в данной задаче моделируется процессами *LineTrafficGenerator*, см. код 1 (полный текст исходных кодов приведён в приложении А). В качестве датчиков, фиксирующих присутствие автомобилей, выступает массив каналов *carsWaiting* — по одному каналу на каждый светофор.

Автомобили генерируются недетерминированно. Для каждого направления создаётся новый процесс *LineTrafficGenerator*, отвечающий за генерацию машин, движущихся через соответствующий светофор.

[1] `http://www.spinroot.com/`

Listing 1: Моделирование внешней среды

```
53 /* Car object */
54 mtype = { CAR };
55
56 /* Cars waiting sign for each traffic light */
57 chan carsWaiting[N_TRAFFIC_LIGHTS] = [1] of { mtype };
58
59 proctype LineTrafficGenerator( byte initTlId )
60 {
61   byte tlId;
62
63   tlId = initTlId;
64
65   do
66   :: carsWaiting[tlId] ! CAR;
67   od
68 }
```

2.2 Моделирование светофоров

Каждый светофор моделируется процессом *TrafficLight*, см. код 2.

При своей работе светофор хранит своё состояние (цвет) в глобальном массиве *tlColor* (строка 115).

Светофор зависит от общих ресурсов — пересечений полос движения: чтобы безопасным образом включить зелёный свет на светофоре, необходимо, чтобы на всех светофорах, пути которых пересекают путь данного светофора, горел красный свет, т. е. необходимо единолично захватить ресурсы — пересечения полос, лежащие на пути данного светофора.

Алгоритм работы процесса светофора следующий. При обнаружении машин на полосе светофора, светофор захватывает лежашие на его пути пересечения полос в порядке увеличения их номеров (захват в такой последовательности обеспечивает отсутствие дедлоков, возникающих из-за циклического захвата ресурсов).

После захвата перекрёстков, светофор переключается в состояние зелёного света и пропускает ограниченный поток машин (представленный сообщением *CAR*).

Далее светофор переходит в состояние красного света и возвращает захваченные ранее ресурсы.

Listing 2: Моделирование светофоров

```
111 /* Traffic lights states */
112 mtype = { RED, GREEN };
113
114 /* Traffic light state */
115 mtype tlColor[N_TRAFFIC_LIGHTS];
116
117 /* Main traffic light process */
118 proctype TrafficLight( byte initTlId )
119 {
120   byte tlId;
121
122   tlId = initTlId;
123
124   assert(tlColor[tlId] == RED);
125
126 endTL:
127   do
128   :: carsWaiting[tlId] ? [CAR] ->
129     /* Cars in queue */
```

```
130
131       /* Lock dependent intersections */
132       if
133       :: tlId == SN ->
134         lockIntersection(0, tlId);
135         lockIntersection(1, tlId);
136         lockIntersection(2, tlId);
137       :: tlId == WE ->
138         lockIntersection(0, tlId);
139         lockIntersection(3, tlId);
140       :: tlId == ES ->
141         lockIntersection(2, tlId);
142         lockIntersection(3, tlId);
143         lockIntersection(4, tlId);
144       :: tlId == NE ->
145         lockIntersection(1, tlId);
146         lockIntersection(4, tlId);
147       fi;
148
149       /* Allow passing */
150     progressPassCar:
151       atomic
152       {
153         printf("MSC: Traffic light #%d: GREEN\n", tlId);
154         tlColor[tlId] = GREEN;
155
156         /* Pass car */
157         /* Note: atomic for easier claim construction */
158         carsWaiting[tlId] ? CAR;
159         printf("MSC: Traffix light #%d: pass cars\n", tlId);
160       };
161
162       /* Forbid passing */
163       atomic
164       {
165         printf("MSC: Traffic light #%d: RED\n", tlId);
166         tlColor[tlId] = RED;
167       };
168
169       /* Release dependent intersections */
170       if
171       :: tlId == SN ->
172         unlockIntersection(2);
173         unlockIntersection(1);
174         unlockIntersection(0);
175       :: tlId == WE ->
176         unlockIntersection(3);
177         unlockIntersection(0);
178       :: tlId == ES ->
179         unlockIntersection(4);
180         unlockIntersection(3);
181         unlockIntersection(2);
182       :: tlId == NE ->
183         unlockIntersection(4);
184         unlockIntersection(1);
185       fi;
186     od;
```

```
187 }
```

2.3 Моделирование пересечений полос движения

Пересечение полос движений моделируется процессом *Intersection*, см. код 3. Каждому пересечению соответствует один процесс.

Захват и высвобождение пересечений реализовано в функциях-макросах *lockIntersection* и *unlockIntersection*, и работают в соответствии со следующим алгоритмом.

Для захвата пересечения в канал *intersectionLockRequests* посылается запрос *LOCK*. Запросы из этого канала последовательно обрабатываются процессом *Intersection*: на каждый запрос отправляется подтверждение *INT* в канал *intersectionLockGranted*, которое означает, что ресурс единолично предоставлен запросившему его светофору. После окончания работы с ресурсом, светофор высвобождает ресурс посылая сообщение *RELEASE* процессу *Intersection*, выдавшему ресурс. При получении сообщения *RELEASE* процесс *Intersection* переходит в исходное состояние.

Последовательная обработка поступающих сообщений решает проблему голодания процессов, при условии слабой справедливости (*Weak Fairness*) в работе модели.

Listing 3: Моделирование пересечений полос движения

```
70  /* Manager messages */
71  mtype = { LOCK, INT, RELEASE };
72
73  /* Intersections lock/release requests queue.
74   * Message contains requestee traffic light identifier.
75   */
76  chan intersectionLockRequests[N_INTERSECTIONS] =
77    [N_TRAFFIC_LIGHTS] of { mtype, byte };
78  chan intersectionLockGranted[N_TRAFFIC_LIGHTS] =
79    [0] of { mtype };
80  chan intersectionReleaseRequests[N_INTERSECTIONS] =
81    [0] of { mtype };
82
83  /* Macro for obtaining intersection resource */
84  #define lockIntersection( intId, tlId )    \
85    intersectionLockRequests[intId] ! LOCK(tlId); \
86    intersectionLockGranted[tlId] ? INT
87
88  /* Macro for releasing intersection resource */
89  #define unlockIntersection( intId ) \
90    intersectionReleaseRequests[intId] ! RELEASE
91
92  /* Intersection resource manager */
93  proctype Intersection( byte initIntId )
94  {
95    byte intId, tlId;
96
97    intId = initIntId;
98
99  endInt:
100   do
101   :: intersectionLockRequests[intId] ? LOCK(tlId) ->
102     /* Handle request */
103     intersectionLockGranted[tlId] ! INT;
104
105     /* Wait for release */
106   progressGiveIntersection:
107     intersectionReleaseRequests[intId] ? RELEASE;
108   od;
```

```
109 }
```

3 Верификация алгоритма

Была проведена верификация алгоритма по следующим критериям:

1. Безопасность: на светофорах с пересекающимися путями никогда не должен одновременно гореть зелёный свет. Данное условие проверялось следующей LTL формулой, представленной в строках 264–268 кода 4.
2. Живость: всегда, при появлении машин на светофоре, светофор когда-нибудь обязательно загорится зелёным светом. Данное условие проверялось формулами в строках 274–277 кода 4.
3. Справедливость: никакой поток машин не должен бесконечно стоять на светофоре. Данное условие было проверено средствами *Spin*, при условии слабой справедливости в работе модели.

Listing 4: Предикаты для верификации алгоритма

```
239 /*
240  * Correctness requirements.
241  */
242
243 /* Car crash accident definition */
244 #define accident_01 (tlColor[0] == GREEN && tlColor[1] == GREEN)
245 #define accident_02 (tlColor[0] == GREEN && tlColor[2] == GREEN)
246 #define accident_03 (tlColor[0] == GREEN && tlColor[3] == GREEN)
247 #define accident_13 (tlColor[1] == GREEN && tlColor[3] == GREEN)
248 #define accident_23 (tlColor[2] == GREEN && tlColor[3] == GREEN)
249
250 /* Car waiting at traffic light definition */
251 #define car_waiting_0 (len(carsWaiting[0]) > 0)
252 #define car_waiting_1 (len(carsWaiting[1]) > 0)
253 #define car_waiting_2 (len(carsWaiting[2]) > 0)
254 #define car_waiting_3 (len(carsWaiting[3]) > 0)
255
256 /* Traffic light is green definition */
257 #define tl_green_0 (tlColor[0] == GREEN)
258 #define tl_green_1 (tlColor[1] == GREEN)
259 #define tl_green_2 (tlColor[2] == GREEN)
260 #define tl_green_3 (tlColor[3] == GREEN)
261
262 /* Safety: Intersecting roads traffic light both never has GREEN state */
263 /*
264  * [] (!accident_01)
265  * [] (!accident_02)
266  * [] (!accident_03)
267  * [] (!accident_13)
268  * [] (!accident_23)
269  */
270
271 /* Liveness: If cars wait on traffic light, then in future traffic light
272  * became GREEN */
273 /*
274  * [] (car_waiting_0 -> <> tl_green_0)
275  * [] (car_waiting_1 -> <> tl_green_1)
276  * [] (car_waiting_2 -> <> tl_green_2)
277  * [] (car_waiting_3 -> <> tl_green_3)
278  */
```

4 Результат работы

В результате данной работы были изучены материалы, представленные в [1] и [2], построена и верифицирована модель контроллера светофоров перекрёстка, получен практический опыт работы с системой автоматической верификации *Spin*.

A Исходный код

Listing 5: Исходный код

```
1  /*
2   *  This file is part of traffic lights model development and verification.
3   *
4   *  Copyright (C) 2010  Vladimir Rutsky <altsysrq@gmail.com>
5   *
6   *  This program is free software: you can redistribute it and/or modify
7   *  it under the terms of the GNU General Public License as published by
8   *  the Free Software Foundation, either version 3 of the License, or
9   *  (at your option) any later version.
10  *
11  *  This program is distributed in the hope that it will be useful,
12  *  but WITHOUT ANY WARRANTY; without even the implied warranty of
13  *  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
14  *  GNU General Public License for more details.
15  *
16  *  You should have received a copy of the GNU General Public License
17  *  along with this program.  If not, see <http://www.gnu.org/licenses/>.
18  */
19
20   /* Variant 9-13:
21    * SN, WE, NE, ES.
22    */
23   /*
24    *                  N
25    *             2
26    *             |         ^
27    *             |         |
28    *             |       --2--------3
29    *               \     /   |
30    *  W              4      |             E
31    *             /     \    |
32    *            /       --1------->
33    *           |          |
34    *     1 -----3---------0------->
35    *           |          |
36    *           v          |
37    *                      0
38    *                  S
39    *
40    */
41
42 #define SN 0
43 #define WE 1
44 #define NE 2
45 #define ES 3
46
47 /* Number of traffic lights */
48 #define N_TRAFFIC_LIGHTS 4
49
50 /* Number of intersections */
51 #define N_INTERSECTIONS  5
52
53 /* Car object */
```

```
54 mtype = { CAR };
55
56 /* Cars waiting sign for each traffic light */
57 chan carsWaiting[N_TRAFFIC_LIGHTS] = [1] of { mtype };
58
59 proctype LineTrafficGenerator( byte initTlId )
60 {
61   byte tlId;
62
63   tlId = initTlId;
64
65   do
66   :: carsWaiting[tlId] ! CAR;
67   od
68 }
69
70 /* Manager messages */
71 mtype = { LOCK, INT, RELEASE };
72
73 /* Intersections lock/release requests queue.
74  * Message contains requestee traffic light identifier.
75  */
76 chan intersectionLockRequests[N_INTERSECTIONS] =
77   [N_TRAFFIC_LIGHTS] of { mtype, byte };
78 chan intersectionLockGranted[N_TRAFFIC_LIGHTS] =
79   [0] of { mtype };
80 chan intersectionReleaseRequests[N_INTERSECTIONS] =
81   [0] of { mtype };
82
83 /* Macro for obtaining intersection resource */
84 #define lockIntersection( intId, tlId )   \
85   intersectionLockRequests[intId] ! LOCK(tlId); \
86   intersectionLockGranted[tlId] ? INT
87
88 /* Macro for releasing intersection resource */
89 #define unlockIntersection( intId ) \
90   intersectionReleaseRequests[intId] ! RELEASE
91
92 /* Intersection resource manager */
93 proctype Intersection( byte initIntId )
94 {
95   byte intId, tlId;
96
97   intId = initIntId;
98
99 endInt:
100   do
101   :: intersectionLockRequests[intId] ? LOCK(tlId) ->
102     /* Handle request */
103     intersectionLockGranted[tlId] ! INT;
104
105     /* Wait for release */
106   progressGiveIntersection:
107     intersectionReleaseRequests[intId] ? RELEASE;
108   od;
109 }
110
```

```
111 /* Traffic lights states */
112 mtype = { RED, GREEN };
113
114 /* Traffic light state */
115 mtype tlColor[N_TRAFFIC_LIGHTS];
116
117 /* Main traffic light process */
118 proctype TrafficLight( byte initTlId )
119 {
120   byte tlId;
121
122   tlId = initTlId;
123
124   assert(tlColor[tlId] == RED);
125
126 endTL:
127   do
128   :: carsWaiting[tlId] ? [CAR] ->
129     /* Cars in queue */
130
131     /* Lock dependent intersections */
132     if
133     :: tlId == SN ->
134       lockIntersection(0, tlId);
135       lockIntersection(1, tlId);
136       lockIntersection(2, tlId);
137     :: tlId == WE ->
138       lockIntersection(0, tlId);
139       lockIntersection(3, tlId);
140     :: tlId == ES ->
141       lockIntersection(2, tlId);
142       lockIntersection(3, tlId);
143       lockIntersection(4, tlId);
144     :: tlId == NE ->
145       lockIntersection(1, tlId);
146       lockIntersection(4, tlId);
147     fi;
148
149     /* Allow passing */
150   progressPassCar:
151     atomic
152     {
153       printf("MSC: Traffic light #%d: GREEN\n", tlId);
154       tlColor[tlId] = GREEN;
155
156       /* Pass car */
157       /* Note: atomic for easier claim construction */
158       carsWaiting[tlId] ? CAR;
159       printf("MSC: Traffix light #%d: pass cars\n", tlId);
160     };
161
162     /* Forbid passing */
163     atomic
164     {
165       printf("MSC: Traffic light #%d: RED\n", tlId);
166       tlColor[tlId] = RED;
167     };
```

```
168
169       /* Release dependent intersections */
170       if
171       :: tlId == SN ->
172         unlockIntersection(2);
173         unlockIntersection(1);
174         unlockIntersection(0);
175       :: tlId == WE ->
176         unlockIntersection(3);
177         unlockIntersection(0);
178       :: tlId == ES ->
179         unlockIntersection(4);
180         unlockIntersection(3);
181         unlockIntersection(2);
182       :: tlId == NE ->
183         unlockIntersection(4);
184         unlockIntersection(1);
185       fi;
186     od;
187   }
188
189   /* The main model function */
190   init
191   {
192     byte tlId, intId;
193
194     /* Reset traffic lights colors */
195     tlId = 0;
196     do
197     :: tlId < N_TRAFFIC_LIGHTS ->
198       tlColor[tlId] = RED;
199       tlId++;
200     :: else ->
201       break;
202     od;
203
204     atomic
205     {
206       /* Start intersection managers processes */
207       intId = 0;
208       do
209       :: intId < N_INTERSECTIONS ->
210         run Intersection(intId);
211         intId++;
212       :: else ->
213         break;
214       od;
215
216       /* Start traffic lights processes */
217       tlId = 0;
218       do
219       :: tlId < N_TRAFFIC_LIGHTS ->
220         run TrafficLight(tlId);
221         tlId++;
222       :: else ->
223         break;
224       od;
```

```
225
226       /* Start cars generator process */
227       /*run CarsGenerator();*/
228       tlId = 0;
229       do
230       :: tlId < N_TRAFFIC_LIGHTS ->
231         run LineTrafficGenerator(tlId);
232         tlId++;
233       :: else ->
234         break;
235       od;
236     }
237   }
238
239   /*
240    * Correctness requirements.
241    */
242
243   /* Car crash accident definition */
244   #define accident_01 (tlColor[0] == GREEN && tlColor[1] == GREEN)
245   #define accident_02 (tlColor[0] == GREEN && tlColor[2] == GREEN)
246   #define accident_03 (tlColor[0] == GREEN && tlColor[3] == GREEN)
247   #define accident_13 (tlColor[1] == GREEN && tlColor[3] == GREEN)
248   #define accident_23 (tlColor[2] == GREEN && tlColor[3] == GREEN)
249
250   /* Car waiting at traffic light definition */
251   #define car_waiting_0 (len(carsWaiting[0]) > 0)
252   #define car_waiting_1 (len(carsWaiting[1]) > 0)
253   #define car_waiting_2 (len(carsWaiting[2]) > 0)
254   #define car_waiting_3 (len(carsWaiting[3]) > 0)
255
256   /* Traffic light is green definition */
257   #define tl_green_0 (tlColor[0] == GREEN)
258   #define tl_green_1 (tlColor[1] == GREEN)
259   #define tl_green_2 (tlColor[2] == GREEN)
260   #define tl_green_3 (tlColor[3] == GREEN)
261
262   /* Safety: Intersecting roads traffic light both never has GREEN state */
263   /*
264    * [] (!accident_01)
265    * [] (!accident_02)
266    * [] (!accident_03)
267    * [] (!accident_13)
268    * [] (!accident_23)
269    */
270
271   /* Liveness: If cars wait on traffic light, then in future traffic light
272    * became GREEN */
273   /*
274    * [] (car_waiting_0 -> <> tl_green_0)
275    * [] (car_waiting_1 -> <> tl_green_1)
276    * [] (car_waiting_2 -> <> tl_green_2)
277    * [] (car_waiting_3 -> <> tl_green_3)
278    */
```

Список литературы

[1] И.В. Шошмина and Ю.Г. Карпов. *Введение в язык Promela и систему комлексной верификации Spin.* СПбГПУ, 2009.

[2] Ю.Г. Карпов. *Model checking. Верификация параллельных и распределенных программных систем.* БХВ-Петербург, 2010.