

Verification of Reactive Systems

#2 Homework Solutions

محمد حسین خوشه چین - ۹۹۲۱۰۱۶۴

۷ اردیبهشت ۱۴۰۰

توضیحات

علاوه بر نوشتن توضیحات مدل های ساخته شده و واریسی آن ها، یک فیلم نیز تهیه شده که در آن بنده با جزئیات بیشتری مدل ها و نحوه واریسی آن ها را توضیح می دهم. برای مشاهده فیلم در اینجا کلیک کنید. برای دانلود فایل کدهای نوشته شده نیز اینجا کلیک کنید. مدل های ساخته شده و ام گرفته از الگوریتم مقاله خانم آقازاده [۲] می باشد.

سوال

الگوریتم هایی مانند الگوریتم پترسون^۱ و یا الگوریتم دکسترا^۲ نمی توانند بر روی یک حافظه به اشتراک گذاشته شده ناشناس^۳ اجرا شوند به طوری که در نهایت مطمئن بود که ویژگی های ایمنی^۴ را ارضا می کنند. آن الگوریتم ها بر روی حافظه های به اشتراک گذاشته شده غیر ناشناس قابل اجرا هستند. در آن حافظه ها یک توافق قبلی^۵ بر روی نام ثبات^۶ های به اشتراک گذاشته شده حافظه وجود دارد. به فرض اگر ثبات های یک حافظه را به شکل یک آرایه مانند $R[1..n]$ در نظر بگیریم که n ثبات دارد، برای هر اندیس مانند x ، نام ثبات $R[x]$ معرف همان ثباتی است که هر پردازش^۷ با آدرس $R[x]$ به آن دسترسی پیدا می کند. وجود یک چنین توافق از پیش تعیین شده ای، پیاده سازی قواعد هماهنگ سازی^۸ ای که پردازش ها باید مطابق آن ها

Peterson's Algorithm^۱

Dijkstra's Algorithm^۲

Anonymous Shared-Memory^۳

Safety^۴

Priori Agreement^۵

Register^۶

Process^۷

Coordination Rules^۸

پیش بروند را برای ما تسهیل می کند به طوریکه ویژگی های ایمنی حفظ شوند. اما در حافظه های به اشتراک گذاشته شده ناشناس چنین توافق قبلی بین پردازش ها بر روی نام ثبات ها وجود ندارد. برای مثال فرض کنید اگر آدرس یک ثبات از دید یک ناظر خارجی^۹ برابر $R[1]$ باشد ممکن است پردازش شماره i با آدرس $R_i[2]$ و پردازش شماره j با آدرس $R_j[3]$ به ثبات مد نظر ناظر خارجی دسترسی پیدا کنند. [۱]

سوال ۱

برای مدل سازی این الگوریتم دو کلاس ری اکتیو به نام های Process و Memory تعریف می کنیم. در شکل ۱ کد کلاس Process و در شکل ۲ کد کلاس Memory نوشته شده است.

^۹External Observer

```

1 reactiveclass Process(10){
2
3   knownrebecs{
4     Memory m;
5   }
6
7   statevars{
8     byte uid;
9     byte [5] mem;
10    byte [5] add;
11    boolean winner;
12  }
13
14  Process(byte u, byte a, byte b, byte c, byte d, byte e){
15    uid = u;
16    for(byte i = 0 ; i < 5 ; i++){ mem[i] = 0; }
17    add[0] = a;
18    add[1] = b;
19    add[2] = c;
20    add[3] = d;
21    add[4] = e;
22    winner = false;
23    self.init(0);
24  }
25
26  msgsrv init(int round){
27    m.receive(uid,round);
28  }
29
30  msgsrv get(byte v, int l){
31    mem[add[l]-1] = v;
32    if(l != 4){
33      m.receive(uid,l+1);
34    }else{
35      byte owner = owned();
36      if(owner < findMostPresent()){
37        //loser = true;
38        m.result(false,true,uid);
39      }else{
40        if(owner == 5){
41          winner = true;
42          self.crit();
43        }else{
44          m.result(false,false,uid);

```

```

45      }
46    }
47  }
48 }
49
50 msgsrv crit(){
51   winner = false;
52   m.result(true,false,uid);
53 }
54
55 msgsrv starter(){
56   for(byte i = 0 ; i < 5 ; i++){ mem[i] = 0; }
57   m.receive(uid,0);
58 }
59
60 byte owned(){
61   byte owner = 0;
62   for(byte i = 0 ; i < 5 ; i++){
63     if(mem[i] == uid) { owner++; }
64   }
65   return owner;
66 }
67
68 byte findMostPresent(){
69   byte [5] temp;
70   byte mostPresent = 0;
71   for(byte i = 0 ; i < 5 ; i++){
72     temp[i] = 0;
73   }
74   for(byte i = 0 ; i < 5 ; i++){
75     for(byte j = 0 ; j < 5 ; j++){
76       if(mem[i] == mem[j]) { temp[i]++; }
77     }
78   }
79   mostPresent = temp[0];
80   for(byte i = 1 ; i < 5 ; i++){
81     if(mostPresent < temp[i]){ mostPresent = temp[i]; }
82   }
83   return mostPresent;
84 }
85 }

```

شکل ۱: کد کلاس Process

```

1 reactiveclass Memory(25){
2   knownrebecs{
3     Process p1;
4     Process p2;
5     Process p3;
6   }
7
8   statevars{
9     byte [5] reg;
10    byte [3] p;
11    byte cntr;
12    byte cv;
13  }
14
15  Memory(){
16    for(byte i=0 ; i<5 ; i++) { reg[i] = 0; }
17    for(byte i=0 ; i<3 ; i++) { p[i] = 0; }
18    cntr = 3;
19    cv = 3;
20  }
21
22  msgsrv receive(byte u, int l){
23    if(reg[l] == 0) { reg[l] = u; }
24    cntr--;
25    if(cntr == 0){
26      cntr = cv;
27      if(p[0] != 1) { p1.get(reg[l],l); }
28      if(p[1] != 1) { p2.get(reg[l],l); }
29      if(p[2] != 1) { p3.get(reg[l],l); }
30    }
31  }
32
33  msgsrv result(boolean w, boolean l, byte u){
34    if(sender == p1){
35      if(w == true){ p[0] = 3; }
36      if(l == true){
37        p[0] = 1;
38        for(byte i=0 ; i<5 ; i++){
39          if(reg[i] == u) { reg[i] = 0; }
40        }
41      }
42      if(w == false && l == false) { p[0] = 2; }
43    }
44    if(sender == p2){
45      if(w == true){ p[1] = 3; }
46      if(l == true){
47        p[1] = 1;
48        for(byte i=0 ; i<5 ; i++){
49          if(reg[i] == u) { reg[i] = 0; }
50        }
51      }
52      if(w == false && l == false) { p[1] = 2; }
53    }
54    if(sender == p3){
55      if(w == true){ p[2] = 3; }
56      if(l == true){
57        p[2] = 1;
58        for(byte i=0 ; i<5 ; i++){
59          if(reg[i] == u) { reg[i] = 0; }
60        }
61      }
62      if(w == false && l == false) { p[2] = 2; }
63    }
64    if(p[0] != 0 && p[1] != 0 && p[2] != 0 ){
65      cv = 0;
66      if(p[0] == 3 || p[1] == 3 || p[2] == 3 ){
67        for(byte i=0 ; i<5 ; i++) { reg[i] = 0; }
68        for(byte i=0 ; i<3 ; i++) { p[i] = 0; }
69        cntr = 3;
70        cv = 3;
71        p1.starter();
72        p2.starter();
73        p3.starter();
74      }
75      else{
76        if(p[0] == 2){ p1.init(0); }
77        if(p[1] == 2){ p2.init(0); }
78        if(p[2] == 2){ p3.init(0); }
79        for(byte i=0 ; i<3 ; i++){
80          if(p[i] == 2){
81            p[i] = 0;
82            cv++;
83          }
84        }
85        cntr = cv;
86      }
87    }
88  }

```

شکل ۲: کد کلاس Memory

کلاس Process دو آرایه دارد یکی به نام mem و دیگری به نام add. اولی برای نگهداری مقادیری است که از خانه های حافظه خوانده است و دومی برای پیاده سازی مفهوم حافظه ناشناس. هر پردازش در ابتدا یک پیام init به خودش می فرستد. با خواندن و اجرای آن پیام رقابت را برای تصاحب خانه های حافظه شروع می کند. حافظه با دریافت پیام receive چک میکند که اگر خانه مورد نظر خالی است مقدار پردازش درخواست دهنده را درون آن قرار دهد. بعد از آن که رقابت همه پردازش ها بر سر آن خانه مورد نظر از حافظه تمام شد به تمام پردازش ها مقدار نهایی آن خانه از حافظه را در قالب پیام get اعلام میکند. هر پردازش با دریافت پیام get مقدار خوانده شده از حافظه را در آرایه محلی اش ذخیره می کند و پیام بعدی را برای رقابت بر سر خانه بعدی می فرستد. زمانی که رقابت بر سر آخرین خانه به پایان رسید بررسی می کند که آیا برنده شده یا خیر. اگر برنده شده بود با ارسال پیام crit به خودش و اجرای آن وارد ناحیه بحرانی می شود و پس از خارج شدن از ناحیه بحرانی برنده شدنش را با پیام Result

به حافظه اطلاع می دهد. اگر پردازنده بازنده بود با ارسال پیام Result به حافظه این موضوع را اعلام می کند. حافظه با دریافت این پیام خانه های اشغال شده توسط آن پردازنده را خالی می کند. اگر پردازنده نه برنده بود و نه بازنده این موضوع را به حافظه با ارسال پیام Result اطلاع می دهد. زمانی که حافظه وضعیت تمام پردازنده ها را دریافت کرد چک می کند که آیا برنده داشتیم یا نه. اگر برنده داشتیم آنگاه تمام خانه ها را خالی می کند و با ارسال پیام Starter به تمام پردازنده ها به آن ها اعلام می کند که از ابتدا برای دور بعدی رقابت شروع کنند. اما اگر برنده نداشتیم به پردازنده هایی که بازنده نیستند با ارسال پیام init اطلاع می دهد که مجدداً بر سر خانه ها رقابت کنند تا بتوانند خانه های خالی را تصاحب کنند. تابع برنامه نیز به قرار شکل ۳ می باشد. حال در ادامه به بررسی خاصیت ایمنی و سرزندگی این مدل می پردازیم. در شکل ۴

```

1 main{
2   Process p1(m):(1,1,2,3,4,5);
3   Process p2(m):(2,2,3,5,1,4);
4   Process p3(m):(3,3,1,4,5,2);
5   Memory m(p1,p2,p3):();
6 }

```

شکل ۳: کد تابع main

تعریف خواص ایمنی و سرزندگی برای این مدل آورده شده است.

```

1 property {
2
3   define{
4     pw1 = p1.winner;
5     pw2 = p2.winner;
6     pw3 = p3.winner;
7
8     mutex1 = !(p1.winner && p2.winner);
9     mutex2 = !(p1.winner && p3.winner);
10    mutex3 = !(p2.winner && p3.winner);
11  }
12
13  LTL{
14    Safty: G(mutex1) && G(mutex2) && G(mutex3);
15    Deadlock: G(F (pw1 || pw2 || pw3));
16    Liveness: F(pw1) && F(pw2) && F(pw3);
17  }
18
19 }

```

شکل ۴: کد خواص مدل

در شکل ۵ نتیجه وارسی خاصیت ایمنی را بر روی مدل نوشته شده می بیند و این خاصیت به درستی ارضا شد.

Attribute	Value
Total Spent Time	1
Number of Reached States	30169
Number of Reached Transi	45727
Consumed Memory	1930816
▼ CheckedProperty	
Property Name	Safty
Property Type	LTL
Analysis Result	satisfied

شکل ۵: خروجی واررسی خاصیت ایمنی

در شکل ۶ نتیجه واررسی خاصیت سرزندگی را بر روی مدل نوشته شده می بیند و این خاصیت به درستی ارضا نمی شود.

Attribute	Value
▼ SystemInfo	
Total Spent Time	0
Number of Reached States	44
Number of Reached Transi	48
Consumed Memory	2816
▼ CheckedProperty	
Property Name	Liveness
Property Type	LTL
Analysis Result	counter example

شکل ۶: خروجی واررسی خاصیت سرزندگی

سوال ۲

برای وارسی عاری بودن مدل نوشته شده از بن بست کافیت که ویژگی default را روی این مدل بررسی کنیم زیرا ویژگی default یکی از ۴ موردی که بررسی می کند وقوع بن بست می باشد. در شکل ۷ نتیجه واررسی خاصیت default را بر روی مدل نوشته شده می بیند و این خاصیت به درستی ارضا شد.

Attribute	Value
▼ SystemInfo	
Total Spent Time	1
Number of Reached States	30169
Number of Reached Transi	45727
Consumed Memory	1930816
▼ CheckedProperty	
Property Name	System default property
Property Type	LTL
Analysis Result	satisfied

شکل ۷: خروجی واررسی خاصیت عدم بن بست

سوال ۳

ویژگی خواسته شده در صورت سوال را تحت عنوان deadlock نوشته ایم و پیاده سازی آن در شکل ۴ مشهود است. در شکل ۸ نتیجه واررسی خاصیت deadlock را بر روی مدل نوشته شده می بینید و این خاصیت به درستی ارضا شد.

Attribute	Value
▼ SystemInfo	
Total Spent Time	6
Number of Reached States	90121
Number of Reached Transi	352665
Consumed Memory	5767744
▼ CheckedProperty	
Property Name	Deadlock
Property Type	LTL
Analysis Result	satisfied

شکل ۸: خروجی واررسی خاصیت deadlock

سوال ۴

تنها تغییری که نسبت به کد اولیه داریم این است که در کلاس Memory زمانی که یک پردازش به Memory اعلام می کند که بازنده است ابتدا خانه هایی که اشغال کرده را خالی می کنیم و سپس مقادیرش را در یک آرایه کمکی به نام Swap در همان خانه هایی که اشغال کرده بود میریزیم. زمانی که یک پردازش برنده شد و Memory خانه هایش را خالی کرد آرایه کمکی را در خانه های Memory کپی می کنیم. کد تغییر یافته در شکل ۹ آمده است.

```

1 reactiveclass Memory(25){
2
3   knownrebecs{
4     Process p1;
5     Process p2;
6     Process p3;
7   }
8
9   statevars{
10    byte [5] reg;
11    byte [5] swap;
12    byte [3] p;
13    byte cntr;
14    byte cv;
15  }
16
17  Memory(){
18    for(byte i=0 ; i<5 ; i++) { reg[i] = 0; }
19    for(byte i=0 ; i<5 ; i++) { reg[i] = 0; }
20    for(byte i=0 ; i<3 ; i++) { p[i] = 0; }
21    cntr = 3;
22    cv = 3;
23  }
24
25  msgsrv receive(byte u, int l){
26    if(reg[l] == 0) { reg[l] = u; }
27    cntr--;
28    if(cntr == 0){
29      cntr = cv;
30      if(p[0] != 1) { p1.get(reg[l],l); }
31      if(p[1] != 1) { p2.get(reg[l],l); }
32      if(p[2] != 1) { p3.get(reg[l],l); }
33    }
34  }
35
36  msgsrv result(boolean w, boolean l, byte u){
37    if(sender == p1){
38      if(w == true){ p[0] = 3; }
39      if(l == true){
40        p[0] = 1;
41        for(byte i=0 ; i<5 ; i++){
42          if(reg[i] == u) { reg[i] = 0; swap[i] = u; }
43        }
44      }
45
46      if(w == false && l == false) { p[0] = 2; }
47    }
48    if(sender == p2){
49      if(w == true){ p[1] = 3; }
50      if(l == true){
51        p[1] = 1;
52        for(byte i=0 ; i<5 ; i++){
53          if(reg[i] == u) { reg[i] = 0; swap[i] = u; }
54        }
55      }
56      if(w == false && l == false) { p[1] = 2; }
57    }
58    if(sender == p3){
59      if(w == true){ p[2] = 3; }
60      if(l == true){
61        p[2] = 1;
62        for(byte i=0 ; i<5 ; i++){
63          if(reg[i] == u) { reg[i] = 0; swap[i] = u; }
64        }
65      }
66      if(w == false && l == false) { p[2] = 2; }
67    }
68
69    if(p[0] != 0 && p[1] != 0 && p[2] != 0 ){
70      cv = 0;
71      if(p[0] == 3 || p[1] == 3 || p[2] == 3 ){
72        for(byte i=0 ; i<5 ; i++) { reg[i] = 0; }
73        for(byte i=0 ; i<5 ; i++){
74          if(swap[i] != 0){ reg[i] = swap[i]; }
75        }
76        for(byte i=0 ; i<3 ; i++) { p[i] = 0; }
77        cntr = 3;
78        cv = 3;
79        p1.starter();
80        p2.starter();
81        p3.starter();
82      }else{
83        if(p[0] == 2){ p1.init(0); }
84        if(p[1] == 2){ p2.init(0); }
85        if(p[2] == 2){ p3.init(0); }
86        for(byte i=0 ; i<3 ; i++){
87          if(p[i] == 2){
88            p[i] = 0;
89            cv++;
90          }
91        }
92        cntr = cv;
93      }
94    }
95  }
96 }

```

شکل ۹: کد تغییر یافته کلاس Memory

پیاده سازی خواص این مدل دقیقاً مطابق شکل ۴ است. در شکل ۱۰ نتیجه وارسی خاصیت ایمنی را بر روی مدل نوشته شده می بینید و این خاصیت به درستی ارضا شد.

Attribute	Value
▼ SystemInfo	
Total Spent Time	22
Number of Reached States	807067
Number of Reached Transi	1273519
Consumed Memory	51652288
▼ CheckedProperty	
Property Name	Safty
Property Type	LTL
Analysis Result	satisfied

شکل ۱۰: خروجی واررسی خاصیت ایمنی

در شکل ۱۱ نتیجه واررسی خاصیت سرزندگی را بر روی مدل نوشته شده می بیند و این خاصیت به درستی ارضا نمی شود.

Attribute	Value
▼ SystemInfo	
Total Spent Time	0
Number of Reached States	44
Number of Reached Transi	48
Consumed Memory	2816
▼ CheckedProperty	
Property Name	Liveness
Property Type	LTL
Analysis Result	counter example

شکل ۱۱: خروجی واررسی خاصیت سرزندگی

سوال ۵

مقایسه خاصیت سرزندگی دو مدل در شکل ۱۲ آمده است.

Attribute	Value
▼ SystemInfo	
Total Spent Time	0
Number of Reached States	44
Number of Reached Transi	48
Consumed Memory	2816
▼ CheckedProperty	
Property Name	Liveness
Property Type	LTL
Analysis Result	counter example

First Model

Attribute	Value
▼ SystemInfo	
Total Spent Time	1
Number of Reached States	44
Number of Reached Transi	48
Consumed Memory	2816
▼ CheckedProperty	
Property Name	Liveness
Property Type	LTL
Analysis Result	counter example

Second Model

شکل ۱۲: مقایسه خاصیت سرزندگی دو مدل

مدل دوم کمی خاصیت سرزندگی را افزایش می دهد بدین معنی که شاسن پردازنده بازنده در دور بعدی کمی بیشتر است . اما چون باز هم امکان دارد که پردازنده برنده مجدد در دور بعدی تمام خانه های خالی باقی مانده را تصاحب کند لذا خاصیت سرزندگی در مدل دوم نیز ارضا نمی شود.

مراجع

- [1] Michel Raynal , Gadi Taubenfeld. Mutual Exclusion in Fully Anonymous Shared Memory Systems. Information Processing Letters, Volume 158, June 2020.
- [2] Zahra Aghazadeh, Damein Imbs, Michel Raynal, Gadi Taubenfeldm Philipp Woelfel. Optimal Memory-Anonymous Symmetric Deadlock-Free Mutual Exclusion. PODC '19: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing, July 2019, Pages 157–166