Verification of Reactive Systems

#5 Homework Solutions - NuSMV

محمد حسین خوشه چین - ۹۹۲۱۰۱۶۴

۳۱ خرداد ۱۴۰۰

توضیحات

برای حل این سوال از ایده های کتاب [۱]، صفحه ۱۹۱ بخش ۲.۳.۳ کمک گرفته شده است.
برای حل این سوال از نرم افزار وارسی مدل NuSMV ورژن ۲.۶ استفاده شده است.

مسئله کشیش و آدم خوار با NuSMV

این مسئله را با دو رویکرد مختلف حل می کنیم. رویکرد اول مان بدین صورت است که در حالت هایمان متغیرهایی برای سمت راست رودخانه در نظر نمی گیریم اما در رویکرد دوم این متغیرها را نیز به حالت هایمان اضافه می کنیم.

رویکرد اول

در این رویکرد ابتدا متغیر هایمان را به همان صورت که در شکل ۱ مشخص شده است تعریف می کنیم.

```
1 MODULE main
2 VAR
3     Missionary : 0..3;
4     Cannibal : 0..3;
5     BoatM : 0..2;
6     BoatC : 0..2;
7     Side : {-1,1};
8
```

شکل ۱

متغیر Missionary و Cannibal برای شمارش تعداد کشیش ها و آدم خوار ها در سمت چپ رودخانه می باشد و میتوانند مقادیری بین ۰ تا ۳ را بپذیرند. متغیر BoatM و BoatC برای شمارش تعداد کشیش ها و آدم خوار هایی است که سوار قایق شده اند. بخاطر اینکه ظرفیت قایق تنها برای ۲ نفر جا دارد، هنگام تعریف، مقادیر ۰ تا ۲ را برای این متغیر ها در نظر می گیریم. متغیر Side که از نوع Enumeration می باشد نشان دهنده آن است که قایق در سمت چپ و یا در سمت راست رودخانه قرار دارد. قرارداد می کنیم که اگر مقدار Side برابر ۱ بود بدین معنی است که قایق در سمت چپ و اگر ۱- بود بدین معنی است که قایق در سمت راست رودخانه قرار دارد.

حال باید نحوه مقدار دهی اولیه این متغیرها و تعریف مقدارشان در حالت بعدی با توجه به مقدارشان در حالت فعلی را تعریف کنیم. برای اینکه می توانیم از عبارت ASSIGN استفاده کنیم. همچنین برای مقداردهی اولیه حالت هایمان میتوانیم از دستور INIT نیز استفاده کنیم و با عبارت های بولی حالت اولیه سیستم را توصیف کنیم. مزیت این روش این است که می توانیم به شکل غیرقطعی و یا با تعریف یک شرط حالات اولیه سیستم را توصیف کنیم. حال برای تعریف حالات اولیه سیستم به صورت شکل ۲ عمل می کنیم.

```
 9 INIT
10     BoatM + BoatC >= 1;
11
12 ASSIGN
13     init(Missionary) := 3;
14     init(Cannibal) := 3;
15     init(Side) := 1;
```

شکل ۲

در خط ۱۳ تا ۱۵ با استفاده از دستور init متغیرهای مربوط به کشیش ها، آدم خوارها و جایگاه قایق را مقدار دهی می کنیم. تعداد کشیش ها برابر با ۳، تعداد آدم خوارها برابر با ۳ و قایق در سمت چپ رودخانه قرار دارد. در خط ۱۰ ام این شرط را تعریف می کنیم که قایق نباید خالی باشد و حداقل یک نفر باید در قایق برای رفتن به آن سوی رودخانه قرار داشته باشد. دلیل این کار آن است که یکی از شروط مسئله را ارضا کنیم که می گفت همیشه باید یک نفر در قایق حضور داشته باشد. حال چون ممکن است بیش از یک نفر برای انتقال انتخاب شود و یا مشخص نیست که از کدام دسته خواهند بود بنابراین باید به شکل غیرقطعی به همین صورتی که مشاهده می کنید این مقدار دهی را تعریف کنیم.

در ادامه باید تعریف کنیم که متغیرهای حالاتمان زمانی که به یک حالت دیگر می رویم به چه شکلی تغییر کنند. دقت کنید زمانی که این عمل را برای متغیری انجام ندهیم آنگاه مقدار متغیر با توجه به نوع و محدوده مقادیرش به شکل غیر قطعی در هر تغییر حالت تغییر می کند. حال همانطور که در شکل ۳ مشخص است در این مسئله از همان دستور ASSIGN برای تعریف

این اعمال استفاده می کنیم .

```
16      next(Cannibal) := Cannibal - (BoatC * Side);
17      next(Missionary) := Missionary - (BoatM * Side) ;
18      next(Side) := Side * -1;
19
```

شکل ۳

همانطور که مشخص است تابع next مقدار متغیر ها را در حالت های بعدی مشخص می کند. مقدار بعدی متغیر مربوط به کشیش ها و آدم خوارها را اینطور محاسبه می کنیم که اگر قایق در سمت چپ رودخانه باشد بدین معنی است که افراد سوار قایق شدند و باید به همان تعدادی که سوار قایق شدند ازشان کم بشود. حالت دیگر این است که قایق در سمت راست رودخانه قرار دارد و افرادی که توی قایق هستند قرار است به سمت چپ رودخانه برگردند و باید مقدارشان را به آن هایی که در سمت چپ رودخانه قرار دارند اضافه کنیم. اینجا دلیل اینکه چرا متغیر Side را دو مقدار به شکل ۱ و ۱- در نظر گرفتیم مشخص می شود. اگر قایق سمت چپ باشد مقدارش ۱ است و طبق فرمول نوشته شده مقدارش کم می شود. اما اگر قایق سمت راست باشد مقدارش ۱- است و طبق فرمول نوشته شده مقدارش افزوده می شود.

حال در انتها باید فرمول LTL را بنویسیم که شروط مسئله را بررسی کند. سپس نقیض آن فرمول را به برنامه مان اضافه کنیم تا اگر مثال نقضی برای این فرمول وجود دارد به ما نشان دهد. آن مثال نقض نشان می دهد که چطور می توانیم این مسئله را حل کنیم. حال برای مسئله مان فرمول LTL به صورت شکل ۴ می نویسیم.

```
20 LTLSPEC
21      ! ( G(BoatM + BoatC >= 1 & BoatM + BoatC <= 2 & BoatM >=0 & BoatC >= 0) & (((Missionary > 0 & Missionary
   < 3) -> (Cannibal = Missionary)) U (Cannibal = 0 & Missionary = 0)) );
22
```

شکل ۴

این فرمول دو بخش دارد. بخش اول یک عبارت Always و بخش دوم یک Until است که با یکدیگر and شده اند. در عبارت Always بررسی می کنیم که همیشه حداقل یک نفر در قایق قرار داشته باشد و تعداد افرادی که در قایق هستند نباید هیچگاه بیشتر از دو نفر باشد. همچنین این شرط را نیز قرار می دهیم که منفی شدن تعداد افراد در قایق معنی ندارد. در عبارت Until نیز می گوییم همیشه تعداد کشیش ها بیشتر از یا مساوی آدم خوارها باشد تا اینکه جفتشان برابر با صفر شوند. دلیل اینکه در این عبارت می گوییم اگر تعداد کشیش ها برابر با ۳ یا صفر نبود آنگاه باید تعداد کشیش ها با آدم خوار ها برابر باشد آن است که نمی توانیم تعداد بیشتری آدم خوار به سمت راست ببریم زیرا با این کار زمانی که یک کشیش به سمت راست برود خرده می شود.

خروجی وارسی این برنامه در شکل ۵ و ۶ قابل مشاهده می باشد.

```
-- specification !( G (((BoatM + BoatC >= 1 & BoatM + BoatC <= 2) & BoatM >= 0) & BoatC >= 0) & (((Missionary > 0 & Missionary < 3) ->
Cannibal = Missionary) U (Cannibal = 0 & Missionary = 0)))    is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    Missionary = 3
    Cannibal = 3
    BoatM = 1
    BoatC = 1
    Side = 1
  -> State: 1.2 <-
    Missionary = 2
    Cannibal = 2
    BoatC = 0
    Side = -1
  -> State: 1.3 <-
    Missionary = 3
    BoatM = 0
    BoatC = 2
    Side = 1
  -> State: 1.4 <-
    Cannibal = 0
    BoatC = 1
    Side = -1
  -> State: 1.5 <-
    Cannibal = 1
    BoatM = 2
    BoatC = 0
    Side = 1
```

شکل ۵

```
  -> State: 1.6 <-
    Missionary = 1
    BoatM = 1
    BoatC = 1
    Side = -1
  -> State: 1.7 <-
    Missionary = 2
    Cannibal = 2
    BoatM = 2
    BoatC = 0
    Side = 1
  -> State: 1.8 <-
    Missionary = 0
    BoatM = 0
    BoatC = 1
    Side = -1
  -> State: 1.9 <-
    Cannibal = 3
    BoatC = 2
    Side = 1
  -> State: 1.10 <-
    Cannibal = 1
    BoatC = 1
    Side = -1
  -- Loop starts here
  -> State: 1.11 <-
    Cannibal = 2
    BoatC = 2
    Side = 1
  -> State: 1.12 <-
    Cannibal = 0
    Side = -1
```

شکل ۶

رویکرد دوم

در این رویکرد ابتدا متغیر هایمان را به همان صورت که در شکل ۷ مشخص شده است تعریف می کنیم.

```
 1 MODULE main
 2 VAR
 3     LM : 0..3;
 4     LC : 0..3;
 5     RM : 0..3;
 6     RC : 0..3;
 7     BM : 0..2;
 8     BC : 0..2;
 9     Boat : {left,right};
10
```

شکل ۷

متغیر LM و LC برای شمارش تعداد کشیش ها و آدم خوار ها در سمت چپ رودخانه می باشد و میتوانند مقادیری بین ۰ تا ۳ را بپذیرند. متغیر RM و RC برای شمارش تعداد کشیش ها و آدم خوار ها در سمت راست رودخانه می باشد و میتوانند مقادیری بین ۰ تا ۳ را بپذیرند. متغیر BM و BC برای شمارش تعداد کشیش ها و آدم خوار هایی است که سوار قایق شده اند. بخاطر اینکه ظرفیت قایق تنها برای ۲ نفر جا دارد، هنگام تعریف، مقادیر ۰ تا ۲ را برای این متغیر ها در نظر می گیریم. متغیر Boat که از نوع Enumeration می باشد نشان دهنده آن است که قایق در سمت چپ و یا در سمت راست رودخانه قرار دارد.

با استفاده از دستور INVAR یک شرط invariant تعریف می کنیم. شرطی که باید در تمام حالات برقرار باشد و آن شرط این است که در قایق همیشه باید یک نفر وجود داشته باشد و افرادی که درون قایق هستند هیچگاه نباید تعدادشان بیشتر از ۲ نفر باشد. این شرط مطابق شکل ۸ در مسئله مان تعریف کردیم.

```
10
11 INVAR
12     BM+BC < 3 & BM+BC > 0 ;
13
```

شکل ۸

حال باید نحوه مقدار دهی اولیه این متغیرها و تعریف مقدارشان در حالت بعدی با توجه به مقدارشان در حالت فعلی را تعریف کنیم. به صورت شکل ۹ عمل می کنیم. این بخش تفاوت چندانی با رویکرد اول ندارد و با همان شکل متغیرها را مقدار دهی کردیم.

```
14 INIT
15     BM+BC > 0;
16
17 ASSIGN
18     init(LM) := 3;
19     init(LC) := 3;
20     init(RM) := 0;
21     init(RC) := 0;
22     init(Boat) := left;
23
```

شکل ۹

در ادامه باید تعریف کنیم که متغیرهای حالاتمان زمانی که به یک حالت دیگر می رویم به چه شکلی تغییر کنند. همانطور که در شکل ۱۰ مشخص است در این مسئله از همان دستور ASSIGN برای تعریف این اعمال استفاده می کنیم .

```
23
24     next(LM) := case
25                     LM!=0 & LM > LC & BM+BC <2 & Boat = left: LM - BM;
26                     Boat = right : LM+BM;
27                     TRUE : LM;
28                 esac;
29
30     next(LC) := case
31                     LC!=0 & RC < RM & BM+BC <2 & Boat = left: LC - BC;
32                     Boat = right : LC+BC;
33                     TRUE : LC;
34                 esac;
35
36     next(Boat):= case
37                     (Boat = left) & BM+BC = 2 : right;
38                     (Boat = right) & BM+BC = 1 : left;
39                     TRUE : Boat;
40                  esac;
41
```

شکل ۱۰

برای محاسبه کردن تعداد کشیش های سمت چپ رودخانه در حالت بعدی از دستور Case استفاده می کنیم. شرط اولی که می نویسیم این است که تعداد کشیش ها مخالف صفر نباشد و تعداد کشیش ها از آدم خوارها بیشتر باشد و قایق در سمت چپ باشد و جا داشته باشد. اگر این شرط ها برابر بود تعداد کشیش هایی که سوار قایق هستند را از تعداد کشیش های سمت چپ رودخانه کم می کنیم. اگر قایق در سمت راست بود آنگاه تعداد کشیش هایی که روی قایق هستند را به تعداد کشیش های سمت چپ رودخانه اضافه می کنیم.

برای محاسبه تعداد آدم خوارها در سمت چپ ابتدا چک می کنیم که تعداد آدم خوار های سمت چپ برابر با صفر نباشد و در سمت راست رودخانه تعداد کشیش ها بیشتر از تعداد آدم خوار ها باشد و قایق در سمت چپ باشد و جای خالی نیز داشته باشد در این صورت تعداد آدم خوار هایی که روی قایق هستند را از تعداد آدم خوارهایی که در سمت چپ رودخانه هستند کم می کنیم. اگر قایق در سمت راست بود آنگاه تعداد آدم خوارهای روی قایق را به تعداد آدم خوارهای سمت چپ رودخانه اضافه می کنیم.

برای جابه جایی قایق نیز ابتدا چک می کنیم اگر قایق سمت چپ بود و هر ۲ ظرفیتش پر شد آنگاه قایق به سمت راست می رود و زمانی که قایق از سمت راست به سمت چپ می رود که حداقل یک نفر درونش باشد.

حال در انتها باید فرمول LTL را بنویسیم که شروط مسئله را بررسی کند. سپس نقیض آن فرمول را به برنامه مان اضافه کنیم تا اگر مثال نقضی برای این فرمول وجود دارد به ما نشان دهد. آن مثال نقض نشان می دهد که چطور می توانیم این مسئله را حل کنیم. حال برای مسئله مان فرمول LTL به صورت شکل ۱۱ می نویسیم.

```
41
42 LTLSPEC
43     !((LM >= LC & RM >= RC) U (LM=0 & LC=0 & RM = 3 & RC =
   3))
```

شکل ۱۱

در این فرمول یک عبارت Until داریم که میگوید همواره تعداد کشیش های سمت چپ بزرگتر یا مساوی تعداد آدم خوارهای سمت چپ باشد و تعداد کشیش های سمت راست همواره بزرگتر یا مساوی تعداد آدم خوارهای سمت راست باشد تا زمانی که به حالتی برسیم که تمام افراد از سمت چپ به سمت راست منتقل شده باشند.

خروجی وارسی این برنامه در شکل ۱۲ و ۱۳ قابل مشاهده می باشد.

```
-- specification !((LM >= LC & RM >= RC) U (((LM = 0 & LC = 0) & RM = 3) & RC = 3))    is false
-- as demonstrated by the following execution sequence
Trace Description: BMC Counterexample
Trace Type: Counterexample
  -> State: 1.1 <-
    LM = 3
    LC = 3
    RM = 0
    RC = 0
    BM = 1
    BC = 0
    Boat = left
  -> State: 1.2 <-
    RM = 2
    BM = 0
    BC = 1
  -> State: 1.3 <-
    LC = 2
    RM = 1
    BM = 1
    BC = 0
  -> State: 1.4 <-
    LM = 2
    RM = 2
    BM = 0
    BC = 1
  -> State: 1.5 <-
    LC = 1
  -> State: 1.6 <-
    LC = 0
    RM = 0
    BM = 1
    BC = 0
```

شکل ۱۲

```
  -> State: 1.7 <-
    LM = 1
  -> State: 1.8 <-
    LM = 0
    RM = 3
    RC = 3
    BM = 0
    BC = 1
```

شکل ۱۳

مراجع

[1] Michael huth , Mark Ryan. Logic in Computer Science Modeling and Reasoning about Systems, 2nd Edition, Cambridge University Press, 2004.