形式化大作业 韩艳东 201328018629009

问题描述

一个船夫、山羊、卷心菜和狼都在河的一边。船夫至多可搭载一位乘客过河。如果船夫离开，山羊会吃掉卷心菜，狼也会吃掉山羊。船夫能否将它们（包括狼、山羊、卷心菜）安全送到河对岸？

问题解决

1)基本思想：这是一个规划问题，但可以用模型检测来解决。我们描述一个迁移系统其中状态代表何种物品在河的哪边。然后寻求从初始状态出发，目标状态是否可达:即是否存在一条从初始状态出发的路径，该路径上有一个状态是所有货物都在河岸的另一侧，而且在想这个状态迁移的过程中，货物不会处于不安全的冲突场合。

所以我们将解决这样两个问题：(a)建立迁移系统；(b)设定目标规则和安全规则。

2)建立迁移系统：

从问题的描述可以看出,在过河的过程中船夫是具有选择权的（这个选择交由计算机执行），他可以选择搭载或者不搭载物品（即狼，羊，菜）。而且，船夫只有和某物品处在同一侧岸边，他才有可能搭载这个物品，并且最多只能搭载一个。这就意味着，若船夫同狼、羊、菜分别位于河的两岸时，就没有办法搭载它们了。

根据以上过河过程的分析，我们可以建立这样一个迁移状态：

船夫：在此岸或者彼岸
山羊：在此岸或者彼岸
狼：在此岸或者彼岸
卷心菜：在此岸或者彼岸

初始时:
    船夫：此岸
    山羊：此岸
    狼：此岸
    卷心菜：在此岸
下个状态：
船夫：在此岸或彼岸 (搭载物品？由计算机在满足条件的物品中随机选择)
山羊：若此时和船夫 同处一岸又被选中，则下个状态在对岸，否则还在此地 。
狼：若此时和船夫同处一岸又被选中，则下个状态 在对岸，否则还在此地。
卷心菜：若此时和船夫同处一岸又被选中，则下个状态在对岸，否则还在此地。

有了上面的分析，我们便可以建模为一个 Nusmv 程序了。每个货物的位置用布尔变量来建模:0 代表物品在初始河岸，1 代表在目标河岸。 因此，ferryman=0 意味着船夫的初始河岸，ferryman=1 表示他在目标河岸。其他变量 goat,cabbage 和 wolf 的情形类似。

3)设定目标规则和安全规则

我们需要周到一条满足 $\phi U \varphi$的路径，其中φ 表示目标状态，ϕ表示安全规则。然后断定是否所有状态满足: $\neg \phi U \varphi$，即没有路径满足 $\phi U \varphi$。这并不是我们期望得到的，我们期望有一条路径不满足$\neg \phi U \varphi$，即有一条达到目标状态且满足安全条件的路径，而且我们期望 Nusmv 能给出这条路径。

NUSMV 代码

```
MODULE main
VAR
    ferryman:boolean;
    goat:boolean;
    cabbage:boolean;
    wolf:boolean;
    carry:{g,c,w,0};
```

```
ASSIGN
    init(ferryman) := FALSE;
    init(goat) := FALSE;
    init(cabbage) := FALSE;
    init(wolf) := FALSE;
    init(carry) := 0;

    next(ferryman) := {FALSE, TRUE};
    next(carry) := case
              ferryman=goat : g;
              TRUE              : 0;
            esac union
            case
              ferryman=cabbage : g;
              TRUE              : 0;
            esac union
            case
              ferryman=wolf : g;
              TRUE              : 0;
            esac   union 0;
    next(goat) := case
              ferryman=goat & next(carry)=g : next(ferryman);
              TRUE                              : goat;
           esac;
    next(cabbage) := case
              ferryman=cabbage & next(carry)=c : next(ferryman);
              TRUE                              : cabbage;
           esac;
    next(wolf) := case
              ferryman=wolf & next(carry)=w: next(ferryman);
              TRUE                              : wolf;
           esac;

LTLSPEC !(( (goat=cabbage | goat=wolf) -> goat=ferryman) U (cabbage
&goat & wolf& ferryman))
```

执行结果

```
-- specification !(((goat = cabbage | goat = wolf) -> goat = ferryman) U
(((cabb
age & goat) & wolf) & ferryman))  is false
-- as demonstrated by the following execution sequence
Trace Description: LTL Counterexample
Trace Type: Counterexample
-- Loop starts here
-> State: 1.1 <-
  ferryman = FALSE
  goat = FALSE
  cabbage = FALSE
  wolf = FALSE
  carry = 0
-> State: 1.2 <-
  ferryman = TRUE
  goat = TRUE
  carry = g
-> State: 1.3 <-
  ferryman = FALSE
  carry = 0
-> State: 1.4 <-
  ferryman = TRUE
  wolf = TRUE
  carry = w
-> State: 1.5 <-
  ferryman = FALSE
  goat = FALSE
  carry = g
-> State: 1.6 <-
  ferryman = TRUE
  cabbage = TRUE
  carry = c
-> State: 1.7 <-
  ferryman = FALSE
  carry = 0
-> State: 1.8 <-
  ferryman = TRUE
  goat = TRUE
  carry = g
-> State: 1.9 <-
  ferryman = FALSE
  wolf = FALSE
  carry = w
-> State: 1.10 <-
```

```
  ferryman = TRUE
  carry = 0
-> State: 1.11 <-
  ferryman = FALSE
  cabbage = FALSE
  carry = c
-> State: 1.12 <-
  ferryman = TRUE
  carry = 0
-> State: 1.13 <-
  ferryman = FALSE
  goat = FALSE
  carry = g
-> State: 1.14 <-
  ferryman = TRUE
  carry = 0
-> State: 1.15 <-
  ferryman = FALSE
```

结果注释: Nusmv 给出了一条无线的路径，我们追求的是在这条路径上有一个状态是我们需要的，既满足 $\phi U \varphi$。但是这个状态之后的事情我们便不管了。

问题提升

调用有界模型检测将产生最短可能路径，也即规划问题的最短最优解。因为模型检测 Nusmv –bmc –bmc_length 7 ferryman.smv 表明在该模型中，LTL 公式成立，这意味着不可能有少于 7 个迁移的解。

```
-- no counterexample found with bound 0
-- no counterexample found with bound 1
-- no counterexample found with bound 2
-- no counterexample found with bound 3
-- no counterexample found with bound 4
-- no counterexample found with bound 5
-- no counterexample found with bound 6
-- specification !(((goat = cabbage | goat = wolf) -> goat = ferryman) U (((cabb
age & goat) & wolf) & ferryman))    is false
-- as demonstrated by the following execution sequence
```

```
Trace Description: BMC Counterexample
Trace Type: Counterexample
-> State: 1.1 <-
  ferryman = FALSE
  goat = FALSE
  cabbage = FALSE
  wolf = FALSE
  carry = 0
-> State: 1.2 <-
  ferryman = TRUE
  goat = TRUE
  carry = g
-> State: 1.3 <-
  ferryman = FALSE
  carry = 0
-> State: 1.4 <-
  ferryman = TRUE
  wolf = TRUE
  carry = w
-> State: 1.5 <-
  ferryman = FALSE
  goat = FALSE
  carry = g
-> State: 1.6 <-
  ferryman = TRUE
  cabbage = TRUE
  carry = c
-> State: 1.7 <-
  ferryman = FALSE
  carry = 0
-> State: 1.8 <-
  ferryman = TRUE
  goat = TRUE
  carry = g
```

人们也许希望验证是否存在设计山羊的三次往返的解，这可以通过修改 LTL 公式来实现。现在我们不去寻找满足 $\phi U \varphi$ 解，代之以寻找满足

$$(\phi U \varphi) \wedge G(goat \rightarrow G\, goat)$$

的路径，这里 ϕ 等于

$$((goat = cabbage \quad goat = wolf) \rightarrow goat = ferryman$$

公式的最后一位说明：一旦山羊过了河，它将继续保持过河，否则山羊至少三次往返。

Nusmv 验证了这个公式的否定是真的，从而确定没有这样的解。

-- specification (!(!(!(goat = cabbage) U !(goat = wolf))) -> goat = ferryman) is true