基于 Alloy 求解过河问题

肖 杨 李国旗
(北京航空航天大学工程系统工程系 北京 100191)

摘 要 Alloy 是一种轻量级的建模语言,其所建模型可以由 Alloy 分析器自动地进行检验,并提供可视化的工具。过河问题是一个经典的逻辑问题,它要求一队人在一定约束下过河到对岸。给出基于 Alloy 的对复杂过河问题求解的代码和可视化的结果,通过对过河问题的基于 Alloy 的求解分析,提出了用 Alloy 语言对复杂问题建模、求解的思路。

关键词 Alloy 过河问题 模型检验

ALLOY BASED SOLUTION FOR COMPLICATED RIVER CROSSING PROBLEM

Xiao Yang Li Guoqi
(*Department of System Engineering of Engineering Technology, Beijing University of Aeronautics and Astronautics, Beijing 100191, China*)

Abstract Alloy is a light weight modelling language. The model set up by it can be checked automatically by Alloy Analyzer and can provide visualization tools. River crossing problem is a classical logic puzzle. In the puzzle, entire team members should cross the river to another bank abiding by some restrictions. In this paper, for the first time, we presented the code of Alloy-based solution for complex river crossing problems and the visualised outcomes. In addition, by the analysis on Alloy-based solutions for river crossing, we presented the thought of modelling and solving complicated problems with Alloy.

Keywords Alloy River crossing problem Model checking

0 引 言

Alloy 分析器[1]是由 MIT(Massachusetts Institute of Technology)的软件设计组开发的,它的主要功能是进行模型检验,是基于模型检验理论的模型分析工具[2]。模型检验是一种形式化验证方法,验证过程通常是通过使用模型检验算法表明特定属性的可满足性来实现的,而这些属性则被形式化为关于系统模型的逻辑方程。Alloy 分析器所使用的建模语言称为 Alloy。Alloy 是一种轻量级的、描述性的、面向对象的结构建模语言。通过使用 Alloy 语言描述模型中的对象特性建立模型,然后 Alloy 分析器可以自动的进行分析,找出模型的反例。

与一般的模型检测工具相比,Alloy 分析器具有以下三个显著的特点:

1) Alloy 分析器应用了"最小范围假设"原理。最小范围假设是指在一些逻辑式中,尽管变量的取值是无限的,但是,如果可以找到一个有限的范围集合来满足条件时,一定有一个最小范围。一个模型如果是变化的,它可能意味着会有无限个状态。应用这个原理之后,通过试探模型的检查范围,可以发现模型反例发生的最小范围。

2) 提供了可视化的工具。当找到模型反例时,它将模型中的对象特性及对象之间关系完整的表示出来,并指出反例路径。分析器找出变化的状态,并将模型反例的状态分步显示,这样更容易读出结果并找到模型缺陷。

3) Alloy 作为一种建模语言,简单易学,没有一些特殊的语义问题。它在对模型中对象进行约束时可以采用一种增量的方式,因此声明对象关系更方便。它的符号使用也比其它大部分模型检验器简洁。

模型检验技术是一种形式化方法,需要极其复杂的数学推导,而 Alloy 分析器将推理自动化,保证其数学验证严谨性基础的同时,解决了繁琐的推理过程,使得设计者无需投入过多的时间检查系统模型,也可以保证系统的严谨和完美。

目前 Alloy 分析器的最新版本是 4.1,该版本在可视化结果演示上清晰易操作。Alloy 可以对软件设计进行建模验证,国外已经有了一些应用,且颇见成效,如《空中交通控制系统》[3]。Alloy 官方网站上有更多的实例介绍,可见其应用前景很好。

过河问题是一个经典的逻辑问题,它要求一队人在一定约束下过河到对岸[4]。这个问题虽然仅仅是在一种假设的前提下进行的,但是十分具有研究价值[5]。过河问题根据复杂程度有多种版本,本文给出了基于 Alloy 的,对过河问题中最复杂的一种形式的求解。另外,通过对简单的过河问题和复杂的过河问题的求解过程,演示了 Alloy 语言的建模、Alloy 分析器对问题的自动求解和结果的可视化等,希望能够以此促进 Alloy 及 Alloy 分析器在国内的普及和应用,起到抛砖引玉的作用。

1 应用 Alloy 求解农夫过河问题

收稿日期:2009 - 06 - 03。肖杨,本科,主研领域:模型检测,软件安全。

1.1 求解简单农夫过河问题

过河问题中比较简单的情况是包括四个对象:农民、狐狸、鸡、谷物。其中包含的约束关系有:

(1) 每次只能有两个对象坐船通过河岸;

(2) 只有农民可以开船;

(3) 当农民不在时,狐狸会吃掉身边的鸡,鸡可以吃掉身边的谷物。

约束(3)是在过河时的变化约束,因此这里应该对这个约束进行一些特殊处理。Alloy 的官方网站上给出了一种解决方法,下面是基于 Alloy 语言的对简单过河问题的求解源代码[6]:

```
open util/ordering[State]
/* 定义农夫和他的财产为 Object. */
abstract sig Object { eats: set Object }
one sig Farmer, Fox, Chicken, Grain extends Object { }
/* 约束当农夫不在时,三个对象吃与被吃的关系. */
fact eating{ eats = Fox -> Chicken + Chicken -> Grain}
/* 定义河的两岸. */
sig State { near, far: set Object }
/* 定义初始状态时,所有对象都未过河. */
fact { first.near = Object && no first.far }
/* 农夫至多只能带一个他的财产过河 */
pred crossRiver [from, from', to, to': set Object] {
  one x: from + {
    from' = from - x - Farmer - from'.eats
    to' = to + x + Farmer
  }
}
/* 约束过河的状态转移关系 */
fact {
  all s: State, s': s.next {
    Farmer in s.near => 
      crossRiver [s.near, s'.near, s.far, s'.far]
    else
      crossRiver [s.far, s'.far, s.near, s'.near]
  }
}
/*规定所有对象都过河时状态转移才能完成. */
run { last.far = Object } for exactly 8 State
```

上述代码完成的功能,依语句次序描述如下:

1) 定义对象集合(Farmer,Fox,Chicken,Grain 作为 Object 的全部子集)、状态集合(near 和 far),定义移动时产生变化要求检验的约束条件(eats);

2) 定义初始状态,所有人都在河的 near 岸;

3) 谓词语句 crossRiver 规定所有对象移动的规则;

4) 约束状态之间转化的规则;

5) 定义最终状态(所有人都在 far 岸),并且要求了检验范围。

这里 fact eating 指的是狐狸、鸡和谷物之间的制约条件。在谓词 crossRriver 中,from 和 to 分别是农民过来的此岸和将去的彼岸的开始状态,而 from′和 to′是农民移动之后两岸的状态,eats 作为对对象的约束,对 from′岸进行约束,使得在农民移动后,来的此岸不会出现"被吃"关系,即"-from′.eats"。而因为在这个简单的过河问题中只有四个对象两种制约关系,因此,农民去的彼岸将来的状态因为有农民在,所以一定不会出现"被吃"关系。这样描述了过河问题中的约束(3)。

模型的谓词语句中出现了 one 这个量词,是指 x 是所有 from 岸上对象中的一个,这个对象可以是农民,也可以是农民之外的对象。当是农民之外的对象时,既是农民同一个其他对象一起过河。当那个对象是农民时,既是指农民自己一个人过河。这样描述了过河问题中的约束(1)。

在 from′和 to′逻辑运算式中,每次移动时,都有农民这个集合参与运算,因此解决了过河问题中的约束 b。

直到代码的最后一行,模型中所有的约束都被描述了,于是求解最后的状态(last.far = Object)是否能够在 8 个状态转换内实现。通过运行 Alloy 分析器可以自动得出过河问题的一个解,如图 1 所示。

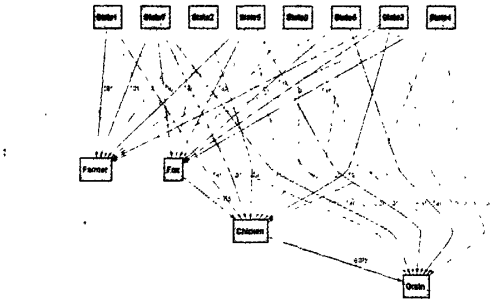

图 1 简单过河问题的解决方法

图 1 中的 State0 表示最初的状态,每个状态通过不同颜色的箭头指向此状态下的对象状态是 far 还是 near,求解的最后的状态是 State7,此时所有的对象都是 far 状态。图 1 列出这个过河方案的具体步骤:

State0:农夫、狐狸、鸡和谷物都在 near 岸;

State1:农夫和鸡在 far 岸,狐狸和谷物在 near 岸;

State2:鸡在 far 岸,农夫、狐狸和谷物在 near 岸;

State3:农夫、鸡和谷物在 far 岸,狐狸在 near 岸;

State4:谷物在 far 岸,农夫、鸡和狐狸在 near 岸;

State5:农夫、狐狸和谷物在 far 岸,鸡在 near 岸;

State6:狐狸和谷物在 far 岸,农夫和鸡仔 near 岸;

State7:农夫、狐狸、鸡和谷物都在 far 岸。

图 1 的图形表示形式是 Alloy 分析器显示检测出的实例的缺省的配置。Alloy 分析器提供了图形定制的功能,常用的还有状态图的表示形式(如图 2 所示)、树形表示形式等。

1.2 求解复杂过河问题

过河问题的元素个数和元素之间约束关系增加都会使问题变复杂。现在以其中一种比较常见的变形进行分析:警察、犯人、父亲、母亲、两个女儿、两个儿子共 8 个对象过河,过河约束是:

(1) 警察、父亲、母亲可以开船,其他人不能开船;

(2) 当警察不在时,犯人会杀死身边的其他人;

(3) 当父亲不在时,母亲会责备身边的儿子;

(4) 当母亲不在时,父亲会责备身边的女儿;

(5) 船上只能承载两个人。

比较两个过河问题可以发现,复杂过河问题除了元素数量外,还增加了复杂的约束关系:原问题中的鸡有吃和被吃两种特性,而父母在拥有被杀和责备两种特性外还拥有开船的资格;谷物原来只被鸡制约,而女儿和儿子同时被犯人和父母所制约;犯人相对于狐狸制约的个体增加了;警察相对于农民只能保证犯人不杀人,却不能保证儿女在父母在的时候的安全(而农民在

时,其他所有对象都是安全的)。

总结这些新的制约后,在建模时需要注意以下几点:

1) 判断 near 和 far 状态是在 from 岸还是 to 岸不能通过某个人来判断(前例中船和农民是绑定在一起的),而需要通过船所在位置判断;

2) 需要选择驾驶员开船,而不是只有一个人可以开船;

3) 约束(2)、(3)和(4)有新的适用条件,且被制约的关系成立条件不一样。

针对这些要求,对上个模型进行修改:

1) 增加一个对象 Boat 来判断每个状态的 from 和 to 的集合,因此在每次移动过程中需要有"-Boat"和"+Boat"操作。因为增加了 Boat 的操作,而 Boat 不能自己移动,如果依照之前的语句只用一条语句,那么就不能找到任何反例了(约束过多),因此这里还需要增加一个移动规则。于是两种移动规则取"并"的关系:一种是只移动一个人 one,一种是移动两个人用 some(some 代表一个或多于一个,在这里实际上只能有两个人随船移动)。

2) 在选取驾驶员的时候,将不能驾驶的人排除,从"from - (Criminal + Girl1 + Girl2 + Boy1 + Boy2 + Boat)"选一个人去驾驶船(one)。因为过河可以是两个人,所以当两个人过河时除了驾驶外,另一个人也可以是驾驶,这通过"one: from - Boat"来解决。

3) 因为对于各个约束约束(2)、(3)和(4)要有不同的适用条件,所以加入一个新的规则来确定过河约束。"pred risking"限制了当警察不在时,约束(2)生效;当父亲不在时,约束(3)生效;当母亲不在时,约束(4)生效。

下面是该模型的代码:

```
/* 签名的声明. */
abstract sig Object{risk1, risk2, risk3: set Object}
one sig Cop, Dad, Mum, Criminal, Girl1, Girl2, Boy1, Boy2, Boat
extends Object {}
/* 约束对象之间的利害关系. */
fact risks{
risk1 = Criminal - >(Dad + Mum + Girl1 + Girl2 + Boy1 + Boy2)
risk2 = Mum - >(Boy1 + Boy2)
risk3 = Dad - >(Girl1 + Girl2)
}
/* 判断利害关系是否成立. */
pred risking[sx: set Object] {
Cop ! in sx = > sx = sx - sx . risk1
Dad ! in sx = > sx = sx - sx. risk2
Mum ! in sx = > sx = sx - sx. risk3
}
/* 定义河的两岸. */
sig State {
near: set Object,
far: set Object
}
/*定义初始状态时,所有对象都未过河. */
fact { first. near = Object && no first. far }
pred crossRiver[from, from', to, to': set Object] {
/* 只有一个人过河时,此人必须会开船. */
(one d: from - Boat -(Criminal + Girl1 + Girl2 + Boy1 + Boy2) {
    {
      from' = from - d - Boat
      risking[from']
      to' = to + d + Boat
      risking[to']
    } ||
/* 两个人过河时,必须有一个会开船. */
(some ds: from - (Criminal + Girl1 + Girl2 + Boy1 + Boy2 + Boat)
| one x: from - Boat |
    {
      from' = from - ds - x - Boat
      risking[from']
      to' = to + ds + x + Boat
      risking[to']
    }
}
/* 过河的状态转移关系. */
fact {
all s: State, s': s. next {
    Boat in s. near = >
    crossRiver [s. near, s'. near, s. far, s'. far] else
    crossRiver [s. far, s'. far, s. near, s'. near]
  }
}
/* 规定所有对象都过河时状态转移才能完成. */
run { last. far = Object } for exactly 18 State
```

通过选择 State 的范围,最终得到结果为 18 个状态是最小反例路径,其中几个状态如图 2 所示(在此限于篇幅只展示 State0、State8,其他状态与此类似)。这些状态图便给出了复杂过河问题一个解的具体步骤。

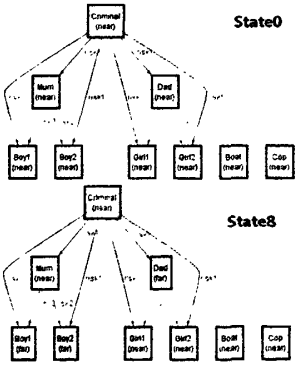

图 2　复杂过河问题解决方案 State0 和 State8

2 总　结

通过对过河问题求解的描述,可知使用 Alloy 语言进行建模,只需要描述清楚约束适用条件,规约谓词以确定下一状态,Alloy 分析器就可以对模型进行自动分析,并生成可视化结果。

在后继的工作中,将应用 Alloy 检验实际的航空软件工程项目的设计和分析模型,并关注基于 CORBA 的中间件系统的设计模型和验证模型的形式化。

参 考 文 献

[1] Daniel Jackson. Alloy: A Lightweight Object Modelling Notation. ACM Transactions on Software Engineering and Methodology (TOSEM'02), 2002, 11(2): 256 - 290.

(下转第 184 页)

(2) 使用绑定 IP 地址的专用预付费帐户进行接口访问的 HTTP 认证;

(3) 详细记录预付费电能管理接口业务操作的审计日志,以便分析业务是否存在异常,追溯业务操作。

预付费电能管理接口主要分两部分:XML 接口和 Web Services 接口。XML 接口提供了根据学(工)号检索系统中所属电表位置信息及开户号信息的功能,是实现第三方预付费管理的基础;Web Services 接口提供了根据开户号操作预付费系统中所属电表的功能,是实现第三方预付费管理的业务整合。从而实现如下四个接口:

(1) 查询当前累计用电量:double getTotal(int accountID);

(2) 查询当前剩余可用电量:double getSurplus(int accountID);

(3) 买入电量:int doCharge(int accountID, double money),接口返回操作结果状态值,如成功、失败、拒绝操作等;

(4) 退还电量:int doUncharge(int accountID, double money)。

2.5 AJAX 和 DHTML 技术实现类桌面应用程序界面

传统 Web 应用的多页面、提交 - 等待 - 刷新机制,使用户得不到立即反馈,交互性很差。在 AJAX[3] 模式下,客户端(浏览器)用户请求由 javascript[4] 代码处理后,在后台以异步方式发送给服务器,而不用等待服务器的响应,用户可以继续输入数据、滚动屏幕和使用系统;服务器只返回结果数据,客户端使用 javascript 代码处理后,立即更新界面,实现桌面应用所具有的动态、快速响应、高交互性的使用体验。

在系统界面组件上,系统采用基于 AJAX 和 DHTML 的组件库 Extjs[5] 开发,使本系统具有了弹出窗口、下拉菜单、右键菜单、拖拽、数据表格、日历框、目录树等桌面应用界面基本元素,提高了用户的使用体验。如图 4 所示,是本系统的售电员售电界面截图,具有和桌面应用基本相同的体验。

图 4 售电员售电界面

3 应用实例效果

系统已在多个高校和市场商铺部署使用,实现了很好的效果,受到了广大用户的欢迎。

1) 系统的实时监控和数据集抄功能,方便用电管理部门对每块电表的剩余电量、总用电量、当前电表状态(如电表开关状态、余电报警状态等)的实时远程监测和定点定时数据集抄,避免了以往人工抄表的繁重工作量和抄写误差。而且这些集抄的基本数据,为用电统计和深层次分析提供了可靠的数据来源,为用电管理部门提供了良好的决策支持和管理支持。如按电表每小时用电量为基础绘制出的 72 小时用电柱状图,可以呈现每块电表的用电状况,在实际使用中,通过该图已为客户查除了多次漏电和用电异常的情况。

2) 系统的自助服务功能,极大地方便了用户的使用,同时也减少了系统的运营和管理成本。如在某高校的使用中,系统完全采用充值卡和自助售电机的自助售电模式,完全取消了售电管理站和售电员售电的模式。师生通过购买充值卡后随时随地上网充电或通过校园一卡通在自助售电机上自动售电,消除了以往售电模式的地域和时间的限制,同时也降低了系统运营的成本。

3) 系统使用 AJAX、DHTML、Comet 等技术,实现了类桌面应用程序的用户界面,提供了友好的操作体验和功能上的人性化设计,如对用户低剩余电量时的自动短信和邮件报警功能。系统在某市场的使用中,低剩余电量报警功能使商铺用户避免了由于异常断电带来的损失,受到用户的欢迎。

4) 系统的平台结构设计和 Web Services 开发接口,使得系统具有了良好的可扩展性。在系统的平台层基础服务之上,已经开发出多个应用程序,同时,在第三方应用整合方面,系统也已与多家一卡通系统进行了成功整合,并在多个高校中实际布置使用。

4 结束语

系统采用基于 TCP/IP 网络的实时通讯与数据采集技术,并结合多种安全机制,实现了随时随地的预付费管理和自助服务;采用结合工业 OPC 技术与 Comet 技术的实时 Web 技术,保证了系统的实时可靠响应;基于 Ajax 技术的系统界面与交互设计,实现了与桌面应用相同的操作体验,为用户的预付费电能管理提供了更完善的解决方案。系统所采用的设计思想和技术方案对类似系统具有参考价值。

参 考 文 献

[1] 全新建,等. 基于 OPC-XML 技术的互联网实时监控系统研究[J]. 自动化仪表,2004,25(11):1 - 4.

[2] Alex Russell. Continuing Intermittent Incoherency[EB/OL]. [2006 - 03]http://alex.dojotoolkit.org/? p=545.

[3] 江恭和,穆斌,武友新,等. 基于 Ajax 技术的文内关键词广告的设计与实现[J]. 计算机应用与软件,2008,25(10):97 - 98.

[4] David Flanagan. JavaScript:The Definitive Guide[M]. 5th ed. O'Reilly,2006.

[5] Extjs. Ext library[EB/OL]. http://extjs.com/.

(上接第 167 页)

[2] Anthony JH Simons, Carlos A Fernandez. Using Alloy to model-check visual design notations[C]//Proceedings of the Sixth Mexican International Conference on Computer Science,2005:121 - 128.

[3] alloy.mit.edu. http://alloy.mit.edu/community/node/227.

[4] 冷明,唐毅. VHDL 语言在状态空间表示法求解过河问题中的应用[J]. 计算机工程,2003,29(6):71 - 73.

[5] 王兆红. 利用图的广度优先搜索解决农夫过河问题[J]. 信息技术,2005(12):102 - 104.

[6] alloy.mit.edu. http://alloy.mit.edu/alloy4/tutorial4/frame-RC-1.html.