Advances in Applied Mathematics 应用数学进展, 2017, 6(9), 1050-1055
Published Online December 2017 in Hans. http://www.hanspub.org/journal/aam
https://doi.org/10.12677/aam.2017.69127

Hans汉斯

Automated Verification of Equivalence of Two Group Definitions Based on Alloy

Ke Zou[1], Jianguo Jiang[1], Yue Xu[1], Xingang Zhang[2]

[1]School of Mathematics, Liaoning Normal University, Dalian Liaoning
[2]No. 24 High School, Dalian Liaoning
Email: zksunny@sina.com, jjgbox@sina.com, xuyuebox@163.com, 52904097@qq.com

Received: Nov. 21st, 2017; accepted: Dec. 4th, 2017; published: Dec. 11th, 2017

Abstract

Group theory is an important part of algebraic system. This paper presents a new method to verify two Group definitions using Alloy. The two definitions of Group are formally described using Alloy, and the automatic verification of the two definitions of Group is realized by the Alloy analyzer. The experimental results show that the method is feasible and has high efficient.

Keywords

Group Theory, Alloy, Formal Description, Automated Verification

基于Alloy的两个群定义的等价性验证

邹　科[1]，江建国[1]，徐　月[1]，张新钢[2]

[1]辽宁师范大学数学学院，辽宁　大连
[2]大连市第二十四中学，辽宁　大连
Email: zksunny@sina.com, jjgbox@sina.com, xuyuebox@163.com, 52904097@qq.com

收稿日期：2017年11月21日；录用日期：2017年12月4日；发布日期：2017年12月11日

摘　要

群论是代数系统中的重要组成部分。本文提出一种使用Alloy验证群论中定理的新方法。使用Alloy对群的两种定义进行了形式化描述，并通过Alloy分析器实现了对这两种群定义的自动化验证。实验结果表明，该方法可行并且具有较高的效率。

文章引用: 邹科, 江建国, 徐月, 张新钢. 基于 Alloy 的两个群定义的等价性验证[J]. 应用数学进展, 2017, 6(9): 1050-1055. DOI: 10.12677/aam.2017.69127

关键词

群论，Alloy，形式化描述，自动化验证

1. 引言

有关群论定理的机器验证一直是自动推理领域内一个基本课题，其研究结果及其方法在代数学、计算机科学、物理等学科中都有重要的应用[1]。

许多学者都在关注着群论定理的机器验证。1993 年，Slaney 使用计算机程序 FINDER 验证了一些半群和拟群的定理[2]。1994 年，张健、Stickel 等人又在计算机上使用 FALCON 解决了几个群论开问题[3]；随后，他们又使用更高效的程序 LDPP 和 SATO 在计算机上解决了多个拟群的开问题[4]。这些工作推动了群论定理机器验证的发展，为机器验证更难的群论开问题提供了一种可行的道路。

FINDER、FALCON、LDPP、SATO 等都属于早期的原型程序，到目前为止都已经没有了后续的工作。这些程序使用的语言表达能力有限。FINDER 由于无法通过消除同构的方式来减小搜索空间，在计算机上运行的时间相对较长，不能高效地处理有关同构的群论定理。FALCON 主要用于验证等式理论，只能应用于部分的群论定理。LDPP、SATO 无法描述包含较长公式的定理[5] [6]。总之，这些工具在验证群论定理的过程中都有一定的缺陷。

本文使用 Alloy 来验证群的两种不同定义的等价性。Alloy 是一个基于关系的一阶语言，表达能力很强，非常适合于描述群论中的定理。它的分析器使用了一种高效的消除对称性(breaking symmetry)算法[7]，这种算法对具有高度对称性的群论定理能显著地提高验证效率。另外，它的分析器能在有限范围内自动查找模型，并能将查找到的模型以图形的方式可视化地显示给用户[8]。我们希望这些优点能使 Alloy 成为一个验证群论定理的利器，可能会对今后群论中开问题或猜想的研究有所帮助。

2. 预备知识

2.1. Alloy 语言及 Alloy 分析器

Alloy 语言是一种轻量级、基于关系的一阶逻辑语言，同时它也是说明性的建模语言。使用 Alloy 语言描述模型非常简洁，而且它可以表示所有数据类型的关系，包括集合、标量和元组，并通过列出属性和约束来描述一个系统的状态和行为，具有极强的描述能力[9] [10]。其所建立的模型可以由 Alloy 分析器进行自动验证，并给出可视化结果。

Alloy 分析器由 MIT 软件设计组开发，是基于模型检验理论的模型分析工具。它将模型转化为布尔表达式，然后使用可满足性理论(SAT Technology)对模型进行分析，进而进行模型检验[11]。Alloy 分析器实际上是一个“模型查找”工具。只要给出一个 Alloy 语言的逻辑表达式，分析器将会在指定的范围内查找符合该逻辑表达式的模型，而这种逻辑表达式就是需要满足的属性。当这种属性不满足时，Alloy 分析器就会产生对应的反例，用于辅助查找为什么会产生该属性不满足的现象。

新版本的 Alloy 分析器使用 Kodkod [12]作为分析引擎。首先将 Alloy 的形式化描述自动转化成 Kodkod API 的 Java 程序，然后再对模型进行求解。Kodkod 擅长对一阶谓词逻辑表示的关系、传递闭包和部分解

的推理计算，因此极大地减少了在计算机上运行的时间。

2.2. 群的概念

群论的主要是研究对象是“群”这个代数结构。我们首先要来了解什么是群，即群的定义是什么。定义一个群有多种不同的方式[13]。

定义 1：设 G 是一个非空集合，如果在 G 中定义了一个二元运算，称为乘法，它满足以下条件，那么称 G 为一个群：

(1) 结合律：$(ab)c=a(bc),a,b,c\in G$；

(2) 存在单位元素：存在$1\in G$，使对任意的$a\in G$，恒有$1a=a1=a$；

(3) 存在逆元素：对任意的$a\in G$，存在$a^{-1}\in G$，使得$a^{-1}a=aa^{-1}=1$。

若将上述条件中的(2)，(3)分别减弱为(2’)，(3’)，则有

定义 2：设 G 是一个非空集合，如果在 G 中定义了一个二元运算，称为乘法，它满足以下条件，那么称 G 为一个群：

(1) 结合律：$(ab)c=a(bc),a,b,c\in G$；

(2’) 存在左(右)单位元素：存在$1\in G$，使对任意的$a\in G$，恒有$1a=a(a1=a)$；

(3’) 存在左(右)逆元素：对任意的$a\in G$，存在$a^{-1}\in G$，使得$a^{-1}a=1\left(aa^{-1}=1\right)$。

3. 基于 Alloy 的群定义等价问题

在数学中，我们已经可以证明定义 1 与定义 2 是等价的。下文我们将通过 Alloy 对群的定义进行形式化描述从而建立模型[14]。再由 Alloy 分析器进行自动分析验证，对得到的可视化结果进行对比分析后，来验证该定理是成立的。

3.1. 构建模型

由定义 1 和定义 2 可以发现，群的定义有三个约束，基于 Alloy 语言，现在建立定义 1 的群的模型(以含有三个元素的群为例)，形式化语言描述如下所示：

```
module Group
sig Element {}
sig Group {
   elements : set Element,
   unit : elements,
   mult : elements -> elements -> one elements,
   inv : elements -> some elements
}{all a,b,c : elements| c.((b.(a.mult)).mult) = (c.(b.mult)).(a.mult)
      /*定义群中元素的结合律*/
      all a : elements | a = unit.(a.mult) && a = a.(unit.mult)
      /*定义群中存在单位元素*/
      all a : elements | unit = a.((a.inv).mult) && (a.inv).(a.mult) = unit
      /*定义群中存在逆元素*/
}
pred AGroup(G: Group) {
      # G.elements = 3
} run AGroup for 3 but 1 Group/*生成一个具有三个元素的群*/
assert Inverse {
      all G: Group, e : G.elements | lone e.(G.inv)
} check Inverse for 5 but 1 Group/*检测群中的逆元唯一性质*/
assert Unit {
      all G: Group, e : G.elements | lone e
} check Unit for 5 but 1 Group/*检测群中的单位元唯一性质*/
```

上述语句的描述及实现的功能如下：

(1) 定义对象集合(Element 及 Group)，定义群中元素的三种关系(unit, mult, inv)；

(2) 描述了定义 1 中的三种约束关系；

(3) 谓词(AGroup)语句，生成一个具有三个元素的群(#在 Alloy 语言中代表个数)；

(4) 断言(Inverse)语句，在规定范围内检测上述群是否具有逆元唯一性质；

(5) 断言(Unit)语句，在规定范围内检测上述群是否具有单位元唯一性质。

定义 2 与定义 1 的区别在于，弱化了定义 1 中的后两个条件，即将“存在单位元”和“存在逆元素”弱化为“存在左(右)单位元”和“存在左(右)逆元素”。因此，在使用 Alloy 语言形式化描述时，只需将描述约束关系的部分进行部分改变即可。再次建立模型，形式化描述语言如下：

```
module Group
sig Element {}
sig Group {
  elements : set Element,
unit : elements,
  mult : elements -> elements -> lone elements,
  inv : elements -> some elements
} { all a,b,c : elements| c.((b.(a.mult)).mult) = (c.(b.mult)).(a.mult)
/*定义群中元素的结合律*/
    all a : elements | a = unit.(a.mult)
    /*定义群中存在单位元素*/
    all a : elements | unit = a.((a.inv).mult)
    /*定义群中存在逆元素*/
}
pred AGroup(G: Group) {
    #G.elements = 3
} run AGroup for 3 but 1 Group/*生成一个具有三个元素的群*/
assert Inverse {
    all G: Group, e : G.elements | lone e.(G.inv)
} check Inverse for 5 but 1 Group/*检测群中的逆元唯一性质*/
assert Unit {
    all G: Group, e : G.elements | lone e
} check Unit for 5 but 1 Group/*检测群中的单位元唯一性质*/
```

3.2. 实验结果

使用 Alloy 分析器分别自动分析上述两个模型，将会得到模型的可视化结果以及分析过程的时间。由结果显示可知，在断言检测中，二者都没有发现反例。在谓词检测中，两个模型都存在着满足约束条件的实例。对比两个实例的可视化结果可以发现，两个实例是一模一样的，即定义 1 与定义 2 生成的群的实例是同一个群。这足以说明二者是是等价的，即原命题得以验证。

如图 1 所示，为生成的具有三个元素的群的实例。

用 *Group* 表示一个群，E_0, E_1, E_2 分别表示群中的三个元素。观察上图中群的实例可知，其中 E_2 表示该群中的单位元(unit)，E_0, E_1 互为逆元(inv)，且满足单位元唯一和逆元唯一的性质。

为说明 Alloy 验证结果的准确性，现在将谓词(Agroup)语句中的# G.elements = 3 语句分别改为# G.elements = 5，# G.elements = 10，# G.elements = 15 等，即增加群中元素的数量。分别在定义 1 和定义 2 的形式化描述下，使用 Alloy 分析器进行验证，生成具有 5 个元素的群，具有 10 个元素的群及具有 15 个元素的群的实例，进而对结果进行对比分析。

如图 2 所示，为生成的具有五个元素的群的实例。

用 E_0, E_1, E_2, E_3, E_4 分别表示群中的五个元素。观察图中群的实例可知，其中 E_4 表示该群中的单位元

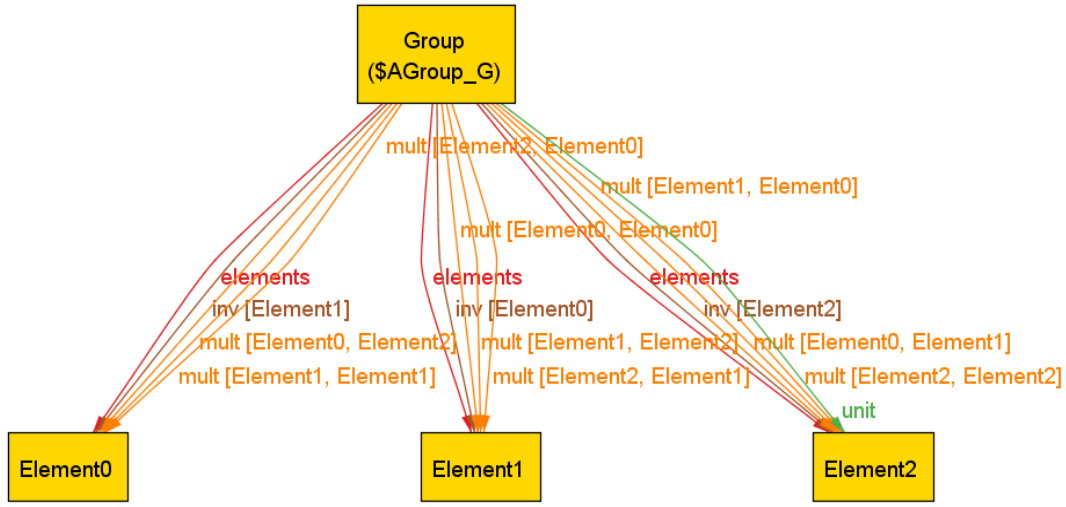

Figure 1. Instance
图 1. 实例

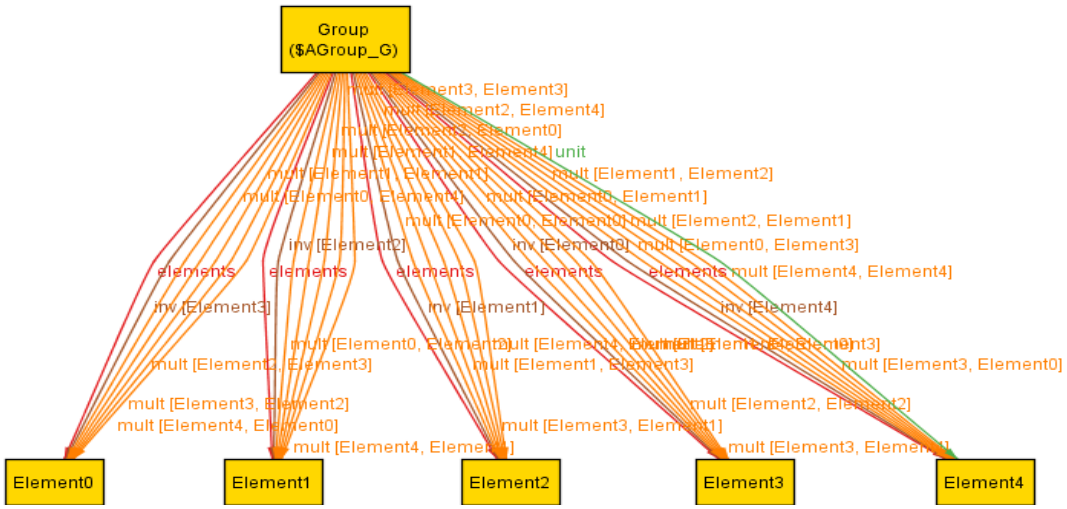

Figure 2. Instance
图 2. 实例

(unit)， E_0, E_1 互为逆元(inv)， E_2, E_3 互为逆元(inv)，且满足单位元唯一和逆元唯一的性质。

对得到的结果进行对比分析，定义 1 和定义 2 生成的具有 10 个和 15 个元素的群同样是相同的。此外，我们还验证了具有更多元素的群，均能得到相同的结果。由此，可证明使用 Alloy 验证群定义是可行的。并且运行的时间虽然随元素的增加而增长，但均能在几分钟甚至几秒内解决。

4. 结论

通过上述对群定义的建模与等价性的验证，可以发现，使用 Alloy 对群定义进行建模是可行的。只需要将问题的约束条件描述清楚，再由 Alloy 分析器对模型进行自动分析，通过对比产生的可视化结果即可将问题解决。并且自动分析模型的时间较短，有效地提高了机器验证的效率。由此可知，Alloy 语言是一种非常有效的形式化分析语言，对群基本问题的分析与验证有着十分显著的作用。希望本文所提出的方法，将对群论中至今未解决的猜想研究起到重要作用，有关使用 Alloy 语言解决群的复杂性问题还有待继续分析研究。

参考文献 (References)

[1] 张禾瑞. 近世代数基础[M]. 北京: 高等教育出版社, 2008.

[2] Slaney, J. (1993) FINDER: Finite Domain Enumerator. Version 3.0 Notes and Guide. Australian National University, Canberra.

[3] Zhang, J. (1996) Constructing Finite Algebras with Falcon. *Journal of Automated Reasoning*, **16**, 1-22. https://doi.org/10.1007/BF00244457

[4] Zhang, H.T. and Stickel, M. (1994) Implementing the Davis-Putnam Algorithm by Tries. Technical Report, University of Iowa, Iowa City.

[5] McCune, W. (1994) A Davis-Putnam Program and its Application to Finite First-Order Model Search: Quasigroup Existence Problems. Mathematics and Computer Science Division, Argonne National Laboratory, Argonne.

[6] Zhang, J. and Zhang, H.T. (1995) SEM: A System for Enumerating Models. In: *Department of Philosophy University of Wisconsin-Madison Mathematics and Computer Science*, 298-303.

[7] Torlak, E. and Jackson, D. (2007) Kodkod: A Relational Model Finder. *Proceedings of the* 13*th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Braga, 632-647. https://doi.org/10.1007/978-3-540-71209-1_49

[8] Jackson, D. (2006) Software Abstractions: Logic, Language, and Analysis. MIT Press, Cambridge, MA.

[9] Jackson, D. (n.d.) Alloy: A Language & Tool for Relational Models. http://alloy.mit.edu/alloy

[10] Jackson, D. (2000) Automating First-Order Relational Logic. *ACM SIGSOFT Software Engineering Notes*, **25**, 130-139. https://doi.org/10.1145/357474.355063

[11] Clarke, E.M. and Emerson, E.A. (1981) Design and Synthesis of Synchronization Skeletons Using Branching-time Temporal Logic. In: *Logic of Programs, Lecture Notes in Computer Science*, Springer, 133: 52-71.

[12] Torlak, E. and Dennis, G. (2006) Kodkod for Alloy Users. Massachusetts Institute of Technology Computer Science and Artificial Intelligence Laboratory, Cambridge.

[13] 聂灵沼, 丁石孙. 代数学引论[M]. 北京: 高等教育出版社, 2011: 23-25.

[14] 杨家海, 姜宁, 安长青, 李福亮. 基于形式化描述的交换机网络自动配置策略的设计与实现[J]. 清华大学学报(自然科学版), 2012(8): 1041-1048.