基于启发式方法的带参系统形式化验证①

段凯强[1,2], 李勇坚[1]

[1](中国科学院软件研究所 计算机科学国家重点实验室，北京 100190)
[2](中国科学院大学，北京 100190)

摘 要： 提出了一种应用启发式信息对带参系统进行形式化验证的新方法。通过引用一个带参系统的小实例，根据其规则和不变式之间的因果关系，可以导出一系列的辅助不变式，从而帮助证明带参系统在任意规模下的正确性。困难之处在于，对于一些复杂的带参系统，难以计算出足够的辅助不变式，这时候可以利用一些启发式信息，减少搜索空间。我们设计并实现了一个验证工具 paraVerifier，并成功利用其证明了 German 和 Flash 等缓存一致性协议在任意规模下的正确性。

关键词： 带参系统；形式验证；不变式查找；启发式方法

Formal Verification of Parameterized Systems based on Heuristics

DUAN Kai-Qiang[1,2], LI Yong-Jian

[1](Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)
[2](University of Chinese Academy of Sciences, Beijing 100190, China)

Abstract： We present a new method for formal verification of parameterized systems based on heuristics in this paper. In order to prove the invariants of a parameterized system with arbitrary nodes, we need to search the corresponding causal relations between the rules and invariants of a parameterized system, discover auxiliary invariants automatically, and construct a formal parameterized proof. The challenge that it's difficult to compute enough auxiliary invariants of complex systems need heuristic information to reduce searching space. We design and implement a tool paraVerifier, and apply our method to some cache coherence protocols successfully, such as German, Flash, etc.

Keywords： parameterized systems; formal verification; invariant computing; heuristics

带参系统的形式化验证是一个有趣而颇具挑战性的问题，因为其具有重要的实践意义。在很多领域中，带参系统都有大规模的实际应用，如缓存一致性协议、安全系统和网络通信协议等。这些带参系统给常规的模型检测技术带来了严峻的挑战。

对带参系统进行形式化验证的困难之处，在于需要保证其安全性质在任意规模的系统中都得到满足。一个系统必须始终遵循其安全性质，也就是说，在这个系统所有能达到的状态中，安全性质保持成立，因而安全性质也称为不变式。对于给定参数的带参系统，也就是带参系统的某个实例，模型检测技术能够自动化验证其不变式，但是无法保证其对带参系统的任意实例都满足。

本文提出了一种对带参系统进行形式化验证的新方法，设计并实现了相应的工具，以类 Murphi[1]语言为前端建模语言，利用启发式信息进行不变式查找，帮助生成定理证明脚本，并成功完成了对一系列缓存一致性协议的验证。本文的创新点主要体现在：

① 我们设计并实现了工具 paraVerifier 对带参系统进行验证。该工具利用类 Murphi 语言作为前端建模语言，简化带参系统的建模过程，方便理解模型，进行调试。

② 从带参系统的小实例出发，利用启发式信息，借助第三方工具进行可满足性和不变式检查，寻找规则和不变式之间的因果关系，并生成足够的辅助不变式。新生成不变式又可以继续帮助生成辅助不变式，直到搜索过程收敛。

① 收稿时间:0000-00-00;收到修改稿时间:0000-00-00

③ 借助找到的规则和不变式之间的因果关系，以及辅助不变式，可以构造形式化证明脚本，并使用定理证明工具 Isabelle 完成证明过程，从而验证带参系统的正确性。

1 相关工作

对带参系统进行验证一直都是形式化领域中的热点和难点。要证明安全性质即不变式在带参系统的所有可达状态下都得到满足，常用的策略有两种，即模型检测技术和定理证明方法。如果使用模型检测技术，需要根据具体系统选用合适的模型检测工具如 Murphi，NuSMV[2]等，建立模型并在有限规模下遍历可达状态集来检测不变式成立与否。但是在更大的规模下，因为可达状态集急剧增长，无法完成验证[3]。为了解决这一问题，产生了其他的验证方法。

Chou C T 等[4]则通过在带参系统中引入抽象节点，以及对迁移规则的前置条件进行强化，逐步迭代得到一个带参系统的抽象模型。一开始的时候通过原始协议得到一个包含抽象节点的抽象模型，抽象模型并不满足原本的不变式，但是仍然提交给模型检测器进行检查，一旦发现反例，就需要仔细地进行人工分析，从而得到一个辅助不变式，进而利用该辅助不变式加强与抽象节点相关的一些迁移规则的前置条件，得到一个加强版的抽象模型，然后周而复始地检测新模型。通过这样的逐步求精过程，最终得到满足原始协议性质的抽象模型。

Lv Y 等[5]则结合参数抽象化、迁移规则前置条件加强和在有限实例中寻找不变式提出了对带参系统进行验证的新方法。他们使用带参协议的一个小实例，即所谓的引用模型，来计算候选不变式。然后将候选不变式中对给定节点的引用抽象化，就可以得到用来加强与抽象节点相关的迁移规则前置条件的强化公式。

Conchon S 等[6]通过实现一种并行反向可达性分析的过程，开发了新的模型检测工具 Cubicle，来验证带参系统的安全性质。他们能够验证的带参系统是基于阵列的，即其带参迁移系统的状态能够使用一系列的数组表示，而这些数组是被任意数目的进程索引的，从而完成对带参系统的任意实例的验证。如果不变式满足，则可以生成便于理解的辅助不变式；反之则生成导致反例的跟踪报告。然而这种方法给出的协议的不变式并不能帮助生成形式化的正确性证明。

曹燊等[7]提出了一种对带参缓存一致性协议进行形式化验证的方法。通过对协议不变式和规则之间可能存在的三种不同的关系进行分析，得到一系列的辅助不变式并完成对带参缓存一致性协议的验证过程，并以 German 协议[8]为例进行了实验。本文主要对其工作进行大量改进，使之能应用于更多的带参系统类型和更大规模的带参系统。

2 模型描述

在我们实现的工具 paraVerifier 中，一个带参系统 S 由一个五元组描述，即 $S = <N, V, I, R, P>$，其中：

① N 是一个字符串，代表系统名称；

② V 是一个变量集合，用于描述带参系统所可能出现的每个状态，也决定状态空间；

③ I 是初始状态集合；

④ R 是规则集合，规则可能是带参规则，由规则名称、规则参数、规则前置条件、规则赋值动作组成，用于描述带参系统各状态间的迁移关系；

⑤ P 是安全性质集合，也就是用来描述带参系统所必须满足的不变式的，可以带参，由性质名称、性质参数、性质内容组成。

paraVerifier 以类 Murphi 语言作为前端建模语言，因此在 paraVerifier 中，模型的表达能力也跟 Murphi 大致相同。Murphi 描述语言是一个高级描述语言，针对状态有限的异步并行系统。Murphi 在某些特性上是高级的，这些特性可以在很多高级编程语言如 Pascal 或者 C 中找到，同时这些特性也是 Murphi 的一部分。例如，Murphi 拥有用户定义的数据类型，过程，以及描述的参数化方法。

我们以 Murphi 描述语言的一个简单但全面的子集作为 paraVerifier 的模型描述语言，使其能够等价地转换到我们的模型描述系统。Murphi 针对的是状态有限的异步并行系统，而我们要验证是在任意规模下带参系统安全性质的正确性。但由于 paraVerifier 需要从一个带参系统的小实例寻找辅助不变式，之后再将其泛化到规模任意的情形，因此这一点并不需要担心。

我们已经实现了一个编译器，用于读入并解析使用 Murphi 描述语言建模的带参系统，转换为等价的内部表示形式，并在此基础上进行进一步的工作。

3 因果关系和不变式查找

3.1 因果关系和一致性引理

规则 $r \in R$ 用于描述带参系统各状态间的迁移过程，其关键组成部分是规则前置条件 *pre*(*r*)，以及规则赋值动作序列 $S = act(r)$。考虑一个公式 *f*，定义算子 *preCond*：

定义 1. $preCond(f, S) = f[x/e]$，其中 $f[x/e]$表示将公式 *f* 中所有变量 *x* 的出现替换为表达式 *e*，而 *e* 是赋值动作序列 *S* 中对变量 *x* 的赋值$\{e \mid x := e, x \in V\}$。

其实 *preCond*(*f, S*)是公式 *f* 对于规则 *r* 的先决条件，也就是如果 *preCond*(*f, S*)成立并且满足规则 *r* 的前置条件 *pre*(*r*)，则执行规则 *r* 的赋值动作序列 *S* 后，公式 *f* 可以得到满足，即：

引理 1. 考虑状态 s_1 和 s_2，s_2 可以从 s_1 通过赋值动作序列 *S* 达到，那么根据 Hoare 逻辑，$s_1 \vdash f$ 当且仅当 $s_2 \vdash preCond(f, S)$。

考虑规则 *r*，公式 *f* 和公式集合 *F*，以及规则赋值动作序列 $S = act(r)$，结合以上定义和引理，我们给出了一些定义：

定义 2. 定义一系列的因果关系 $invHoldForRule_1$，$invHoldForRule_2$，$invHoldForRule_3$ 和 *invHoldForRule*，具体分别为

① $invHoldForRule_1$：$pre(r) \rightarrow preCond(f, S)$成立，也就是说在执行规则 *r* 后，*f* 立即得到满足；

② $invHoldForRule_2$：赋值动作序列 *S* 不改变公式 *f* 中的任何变量，公式 *f* 自然满足；

③ $invHoldForRule_3$：存在公式 $g \in F$，使得能够成立$(g \wedge pre(r)) \rightarrow preCond(f, S)$，其中 *g* 即为一个辅助不变式；

④ *invHoldForRule*：三种关系至少成立其一，即 $invHoldForRule_1 \vee invHoldForRule_2 \vee invHoldForRule_3$。

考虑带参系统的变量集 *V*，其初始状态可以用一系列的公式进行描述，表示其状态变量在初始状态所满足的条件 $inis = \{ini \mid ini$ 描述初态 $x := e$，$x \in V\}$。

定义 3. 对不变式集 *invs*，初始状态条件集 *inis* 和规则集 *R*，如果他们的关系满足以下条件，则称为一致性关系：

① 对任意不变式 *inv*，任意初始状态条件 *ini* 和任意可达状态 *s*，*s* 如果满足 *ini* 则满足 *inv*；

② 对任意不变式 *inv*，任意规则 *r* 和任意可达状态 *s*，关系 *invHoldForRule* 成立。

引理 2. 对某个带参系统，如果其不变式集 *invs*，初始状态条件集 *inis* 和规则集 *r* 满足一致性关系，那么对可达状态集中的每个状态，*invs* 中的每个不变式都能够得到满足[7]。

引理 2 就称为一致性引理。

3.2 不变式查找

在对带参系统进行形式化验证的过程中，最重要的是进行辅助不变式的查找以确定带参系统满足一致性引理。如果出现关系 $invHoldForRule_3$，则必然存在一个新的辅助不变式，我们可以通过某种方法发现这个不变式并继续查找。这个查找过程是一个典型的搜索过程，可以选用广度优先搜索或者深度优先搜索。以广度优先搜索为例，查找辅助不变式的算法如图 1 所示。

```
Input: initial invariants F, protocol instance P = (I, R)
Output: invariant set A, causal relation set C
Process:
A ← F
C ← empty
queue q ← all invariants in F
while q is not empty:
  f ← q.dequeue()
  for every rule r in rule set R:
    rs ← semiPerm(r)
    for every rule instance ri in rs:
      g, rel ← access(ri, f)
      collect new relation rel to C
      if auxiliary invariant g is new:
        collect g to A
        q.enqueue(g)
return A and C
```

图 1.辅助不变式查找算法

这里使用到了两个辅助算法，其中 *semiPerm* 使用启发式策略生成合适规模的规则实例，从而减少搜索空间；*access* 判断不变式 *f* 和规则实例 *ri* 之间属于哪种因果关系。

3.2.1 启发式辅助算法 *semiPerm*

在很多带参系统中，其参数具有对称性，例如在缓存一致性协议中，每个节点都是被同等对待的。基于此考虑，在考虑规则实例的时候就可以使用对称规

约减少搜索空间，例如，假设不变式和规则都是关于一个参数的，那么只需要考虑两个规则实例，即分别与不变式有相同或者不同的参数。该算法如图 2 所示。

```
Input: parameter count of rule r and invariant f
Output: instants of rules
Process:
  n ← parameter count of rule r
  m ← sum of n and parameter count of invariant f
  if m = 0:
    do not generate any rule instance
  else if n = 0:
    return the 0-parameter rule immediately
  else:
    l ← n permutation of number list 1 to m
    for parameter instance p in l:
      remove its other equal parameters from l
    return rules instantiated by l
```

图 2. 规则实例化策略

3.2.2 因果关系判断辅助算法 *access*

该算法直接根据一致性引理进行判断，如果不变式 *f* 和规则实例 *ri* 之间的因果关系是 $invHoldForRule_1$ 或者 $invHoldForRule_2$，则直接返回这个因果关系，不产生新的辅助不变式；否则，除了返回因果关系 $invHoldForRule_3$ 之外，还需要确定新的辅助不变式的精确形式。在这个过程中需要进行一系列的可满足行和不变式正确性检查，需要使用第三方工具 Z3[9]，NuSMV 和 Murphi。为了提高调用效率，以及方便多任务并行运行且能够共享缓存，我们将对第三方工具的调用封装为服务，架构如图 3 所示。

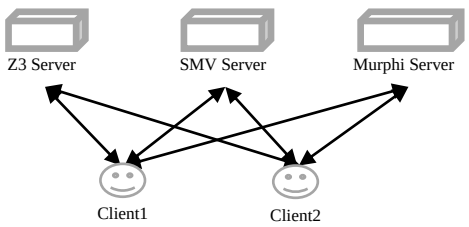

图 3. 第三方工具调用框架

在使用 NuSMV 判断辅助不变式的正确性时，需要先从带参系统小实例对应的 SMV 模型计算出可达状态集，但有时候所需要的小实例会超出可达集能够计算出的范围。为了解决这个问题，我们采用 Murphi 试探性地对不变式进行检查，并缓存结果，取得了良好的效果。

3.3 因果关系的层次

通过图 1 所示的算法，不难发现，在搜索过程中，所发现的因果关系是有明显层次的，具体表现为：

① 对每个不变式，都需要寻找与之相关的关系；

② 对每个规则，都需要进行合适的实例化；

③ 对每个规则实例，都需要与当前不变式进行比对以确定因果关系，实际上这里还有一个隐含的层次，因为规则实例中可能会有分支语句。

4 形式化定理证明生成

Isabelle 是英国剑桥大学和慕尼黑工业大学联合开发的一种通用定理证明工具，为证明系统开发提供了一个通用框架[10]。借助使用 paraVerifier 对带参系统所建立的模型，以及查找到的因果关系和不变式，基于一致性引理，可以生成 Isabelle 证明，从而在 Isabelle 下自动完成对带参系统正确性的验证。

生成的 Isabelle 证明脚本由以下几部分构成：

① 基础部分 paraTheory，主要包含模型所使用的数据结构的定义和基础引理的证明；

② 关于特定带参系统的定义部分，包括枚举，规则和规则集，不变式和不变式集，还有初始状态所满足的条件及其集合；

③ 每个不变式分别关于每条规则的一致性引理的满足情况，即从部分的角度对因果关系的类型进行分析证明；

④ 每个不变式关于规则集的一致性引理满足情况，即从整体的角度进行证明；

⑤ 对每个不变式在初始状态的满足情况进行讨论，而在这里初始状态是由公式描述的，即初始赋值集合会转换为一个公式集合，因此讨论的时候是针对该公式集合进行讨论的；

⑥ 整合以上所有的情况，给出顶层证明脚本，即主定理。

因为对于大型带参系统，生成的证明脚本巨大，甚至无法一次性导入 Isabelle 定理证明工具中，因此我们对证明脚本按照功能不同进行了拆分，先完成对每个部分的证明，最后再从顶层完成全部证明。这一过程可以使用 Isabelle 的 session 功能，编写相应的 ROOT 文件做到。

5 实验结果及分析

我们在四核 Intel Xeon 处理器，8GB 内存，64 位 Linux 3.15.10 环境下对一系列的带参缓存一致性协议进行了实验，包括 MESI，MOESI，Germanish，German 和 Flash，其中 Flash 分带数据通路和不带数据通路两种情况进行讨论。实验数据如表 1 所示，其中带*号的是不带数据通路的 Flash 协议的实验结果。

表 1. 一些带参系统的实验结果

	Rule#	Inv#	Time(s)	Memory(MB)
MESI	4	3	3.33	148
MOESI	5	3	3.34	147
Germanish	6	3	3.47	147
German	13	52	48.20	158
Flash*	60	152	325.60	178
Flash	62	162	589.23	178

其中 MESI，MOESI 和 Germanish 都是规模非常小的带参协议，体现在规则和不变式的数目都比较少，因此运行速度也很快。而 German 协议是一个中型的带参缓存一致性协议，规则数目已经比较多，查找到了较多的不变式，运行时间明显变长。Flash 协议是一个达到工业应用级别的大型带参缓存一致性协议，规则数目庞大，查找到了极多的不变式，且运行时间也比其他案例要长得多，然而这一难题已经被我们成功攻克。从数据中也不难发现，随着带参系统复杂程度的增长，对带参系统进行形式化验证的困难也是急剧增长的。

此外，对每个案例，paraVerifier 最终的输出都是 Isabelle 证明脚本。我们已经把所有生成的证明脚本放在网上[11]，且都在 Isabelle 2015 下运行通过。

6 结论

带参系统的形式化验证具有重要的实践意义，完成验证工作也是一个极大的挑战。我们提出了应用启发式信息对带参系统进行形式化验证的新方法，设计并实现了带参系统验证工具 paraVerifier，从带参系统的小实例出发，利用启发式信息，借助第三方工具进行可满足性和不变式检查，寻找规则和不变式之间的因果关系，并查找到足够的辅助不变式，借此生成 Isabelle 证明脚本，从而完成验证。我们以一系列的带参缓存一致性协议作为研究案例，并成功完成验证。

参考文献

1 Dill DL. The Murphi verification system. Computer Aided Verification. Springer Berlin Heidelberg, 1996: 390–393.

2 Cimatti A., Clarke E, Giunchiglia F. NuSMV: a new symbolic model verifier. Computer Aided Verification, 1999: 495–499.

3 周琰. Godson-T 缓存一致性协议的 Murphi 建模和验证. 计算机系统应用, 2013, 22(10): 124－128.

4 Chou CT, Mannava PK, Park S. A simple method for parameterized verification of cache coherence protocols. Formal Methods in Computer-Aided Design, 2004: 382–398.

5 Lv Y, Lin H, Pan H. Computing invariants for parameter abstraction. Formal Methods and Models for Codesign, 2007. MEMOCODE 2007. 5th IEEE/ACM International Conference on. IEEE, 2007: 29-38.

6 Conchon S, Goel A, Krstić S, et al. Cubicle: A parallel SMT-based model checker for parameterized systems. Computer Aided Verification. Springer Berlin Heidelberg, 2012: 718-724.

7 曹燊, 李勇坚. 基于不变量查找的 German 协议验证. 计算机系统应用, 2015, 24(11): 173-8.

8 German SM. Tutorial on verification of distributed cache memory protocols. Formal Methods in Computer-Aided Design, 2004: 1–77.

9 De Moura, Leonardo, and Nikolaj Bjørner. Z3: An efficient SMT solver. Tools and Algorithms for the Construction and Analysis of Systems. Springer Berlin Heidelberg, 2008: 337-340.

10 Paulson L, Nipkow T, Wenzel M. The Isabelle System. http://isabelle.in.tum.de. 2015.

11 https://github.com/paraVerifier/paraVerifier/tree/master/proof_scripts/