基于不变量查找的 German 协议验证①

曹 桑[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: 提出了一种通过查找缓存一致性协议不变量来验证带参协议正确性的新方法. 缓存一致性协议验证的难点在于必须证明协议对于任意大小的带参系统都成立. 我们通过寻找不变量和协议规则之间的对应关系来计算辅助不变量, 从而帮助推导验证缓存一致性协议. 我们设计实现了一个不变量查找工具并将该工具应用到 German 协议上计算它们的辅助不变量并成功地验证了协议的安全性质.

关键词: 缓存一致性协议; 带参系统; 不变量查找; 多核处理器

Verification of German Cache Coherence Protocol by Searching Invariants

CAO Shen[1,2], LI Yong-Jian[1]

[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 computing invariants for cache coherence protocol in this paper. The verification of cache coherence protocols is always a challenge as we have to check the safety properties of cache coherence protocol with arbitrary nodes. By searching the relations between the invariants and the rules of cache coherence protocol, we can verify the cache coherence protocol with circular reasoning. Besides, we implement a tool to help us computing invariants for cache coherence protocol. We also have experiments on how to apply our method to the German protocol with both control property and data property.

Key words: cache coherence protocol; parameterized system; invariant computing; multi-processor

1 引言

随着计算机技术的日益发展, 多处理器架构在计算机系统结构中出现得愈加频繁. 带参并发系统如缓存一致性协议, 安全协议, 通信协议和网络协议等, 通常由少数异构进程和 N 个同构进程组成, 系统规模 N 就是带参并发系统的参数.

带参系统在实际应用中十分常见. 从计算机中的多处理器架构到工业控制器, 带参系统应用广泛. 缓存一致性协议是一种典型的带参系统. 带参并发系统的验证问题一直是学术界和工业界研究的热点问题. 带参验证的难点在于: 对于给定参数的系统, 我们可以证明其正确性, 但是无法验证系统在任意参数下都是正确的.

本文提出了一种基于不变量查找和定理证明的验证方法, 设计实现了相应工具并对缓存协议进行建模验证, 取得了一定的进展. 本文的创新点主要体现在以下三个方面:

(1) 我们设计实现了辅助不变量查找工具 invFinder 并将该工具应用到缓存一致性协议的验证问题上. invFinder 从缓存协议的小型参数实例出发, 生成辅助不变量来帮助验证协议正确性.

(2) 查找得到的辅助不变量, 不变量和协议规则之间的对应关系可以用来构造形式化证明. 我们使用定理证明工具 Isabella 对自动生成的证明脚本进行验证, 从而说明带参系统的正确性.

(3) 我们将该方法应用到常用的带参协议 German 协议, 成功说明了带参协议的正确性.另外, 我们通过查找得到的辅助不变量来分析协议的正确性, 进一步

① 收稿时间:2015-03-10;收到修改稿时间:2015-04-17

帮助我们理解协议设计.

目前, 人们已经设计了一些缓存一致性协议, 但这些协议没有被形式化证明. 缓存一致性协议是一种带参系统, 证明其在任意规模下的正确性一直是学术界和工业界的热点. 本文提出了一种通过查找缓存一致性协议不变量来验证协议正确性的新方法, 我们成功地将该方法用于验证 German 协议的正确性.

2 相关工作

缓存一致性协议验证的困难在于它是一个带参系统验证, 需要证明安全性质在任意规模下的系统都成立. 常用的验证方法有模型检测方法和定理证明方法. 基于模型检测的方法可以使用模型检测工具如 Murphi[1], NuSMV[2]等验证有穷状态系统并找出不满足系统性质的反例, 但当遇到大规模带参系统时无法继续验证[3,4]. 下面介绍一些主要的验证方法:

Seungjoon Park 和 David L. Dill 运用 PVS 定理证明工具实现了一个 Flash 协议抽象模型[5]. 在这个模型里, 所有缓存状态在单个原子操作中被全局更新. 通过使用抽象函数计算系统当前执行的已提交(committed)事务产生的系统状态来完成证明. 在证明过程中同样需要构造新的系统不变量来继续证明过程, 但这些新构造的不变量会非常复杂而且是整个证明过程中最耗时的部分.

K. L. McMillan 利用组合证明(compositional proof)[6], 时序实例分割(temporal case splitting), 对称性规约(symmetry reduction)等方法和组合模型检测工具 Cadence SMV[7] 形式化证明了参数为 N 的 flash 缓存一致协议在安全性和活性方面的性质[8,9]. 该方法首先构造一个简单的参考模型, 然后通过四个辅助引理来完成证明, 辅助引理用来排除系统在抽象过程中引入的反例.

Ching-Tsun Chou 等提出了一种基于参数抽象和卫士增强的方法, 这种方法先取同构进程中的两个进程, 然后抽象其他进程构造出抽象模型, 最后对得到的新抽象模型检测, 通过反例构造逐步改进模型. 如果当前得到的抽象模型不满足要验证的性质就找出“非干涉引理” 来增强卫士命令[10]. 这种方法依靠人工寻找非干涉引理来增强卫士命令从而改进抽象模型.

Sylvain Conchon 等设计了一种新算法 BRAB 自动计算缓存协议不变量[11,12]. BRAB 算法首先前向搜索固定进程数目的有限系统实例, 然后它对带参系统使用后向可达性分析. BRAB 在每次迭代中计算后向状态的近似状态, 如果它们不在有限状态实例的前向可达集合中, 可以作为候选不变量与原系统中的安全性质一起运用模型检测方法. 如果在执行过程中因为引入不合适近似导致错误路径, BRAB 算法必须回溯路径来保证完备性.

3 German 协议描述

German 协议是由 IBM T.J.Watson 研究中心的 Steven M. German 提出的基于目录结构的缓存一致性协议[13]. German 协议被用在共享内存的并发多处理器系统中以维护缓存一致性.

缓存一致性协议一般分为基于总线侦听的缓存一致性协议和基于目录的缓存一致性协议.[14]基于总线侦听的缓存一致性协议具有低缓存处理延时和结构设计简单等优点.基于目录的缓存一致性协议建立了一个目录来保存和跟踪每个缓存块的状态. 每个缓存控制器将缓存请求直接发向目录, 然后由目录根据当前所有缓存块的状态来决定相应的缓存操作.

在 German 协议中有一个维护目录的主节点 Home 和其他相同的用户节点 Client. 主 Home 节点与用户 Client 节点之间通过消息通道传递消息.

消息类型分为 4 种:

Type1: Client 节点向 Home 节点发送请求(Request to Home node);

Type2: Home 节点向 Client 节点发送使无效消息(Invalidate a remote cache);

Type3: Client 节点向 Home 节点发送无效应答消息(Invalidate Acknowledgement);

Type4: Home 节点向 Client 节点发送同意回复(Grant from Home node).

节点之间的 FIFO 消息通道分为三种:

Channel1: 处理 Client 节点向 Home 节点发送请求;

Channel2: 处理 Home 节点向 Client 节点发送使无效消息和 Home 节点向 Client 节点发送同意回复;

Channel3: 处理 Client 节点向 Home 节点发送无效应答消息.

三种消息通道类型可以避免死锁而且将 Home 节

点发向 Client 节点的消息类型 Type2 和 Type4 放在 Channel2 通道中可以保持一致性. 每个 Client 节点都可以向 Home 节点发送共享(Shared)请求或者独占(Exclusive)请求.

4 协议不变量查找与验证

4.1 不变量查找

对缓存一致性协议的描述通常由以下几个部分组成: 协议变量和常量的定义; 转移规则集合; 初始状态; 不变量集合. 缓存一致性协议的行为部分是一系列转移规则, 每条转移规则由表示条件的卫士命令和修改协议变量的赋值语句组成. 缓存一致性协议可以看作一个状态转移过程, 每个状态就是协议变量对应的赋值. 从初始状态出发, 随着协议规则的执行可以得到一些新的状态, 所有状态的集合组成可达状态集合. 假设状态公式 f 对于可达状态集合中的任意状态都成立, 那么公式 f 是协议的不变量.

在验证工业界的结构复杂的缓存一致性协议时, 证明公式 f 在协议的任意一个可达状态下都成立通常是困难的, 因此我们需要找到其他辅助公式 g, 通过证明 g 在协议状态 S 下成立来推导出公式 f 在协议状态 S 下也成立. 假设有状态公式 f 和转移规则 $t = (guard, statement)$, 其中 $guard$ 是协议转移规则的卫士命令, $statement$ 是协议转移规则对协议变量的修改. 如果状态公式 f 在转移规则 t 执行后的协议状态 S下成立, 我们可以找到状态公式 f 的先决条件 $precond(f, t) = f[e/x]$. 其中, $f[x/e]$ 表示将状态公式 f 中的协议变量 x 替换成表达式 e. 表达式 e 是协议转移规则 t 的 $statement$ 部分对于协议变量 x 的赋值$\{e \mid x:= e, x \in V\}$. 如果先决条件 $precond(f, t)$ 成立并满足协议规则 t 的卫士命令 $guard$, 那么状态公式 f 在协议规则 t 执行后也成立. 相反地, 如果状态公式 f 在协议规则 t 执行后不成立, 我们也能找到对应的先决条件.

定义 1. 对于缓存一致性协议的状态公式 f 和协议转移规则集合中的转移规则 t, 定义如下三种关系:

InvHoldForRule1: 转移规则 t 的赋值部分 $statement$ 不改变状态公式 f 中的协议变量;

InvHoldForRule2: 转移规则 t 的卫士命令 $guard$ 和状态公式 f 的先决条件 $precond(f, t)$之间满足 $(guard => precond(f, t))$ 为真;

InvHoldForRule3: 对于转移规则 t 和状态公式 f, 存在公式 g 使得$((g \& guard) => precond(f, t))$ 为真.

定理 1. 如果状态公式 f 和协议转移规则集合中的任意转移规则 t 之间都存在以上三种关系之一, 而且状态公式 f 在协议的初始状态成立, 那么协议状态公式 f 对于协议可达状态集合中的任意状态 S 也成立.

证明: 假设状态公式 f 与转移规则 t 之间存在关系 *InvHoldForRule1*, 转移规则 t 没有改变协议规则的状态变量, 那么公式 f 在规则 t 执行后的状态下成立; 假设状态公式 f 与转移规则 t 之间存在关系 *InvHoldForRule2*, 公式 f 的先决条件满足规则 t 的卫士命令, 那么状态公式 f 在规则 t 执行后也成立; 假设状态公式 f 和转移规则 t 之间存在关系 *InvHoldForRule3*, 根据先决条件 $precond(f, t)$ 和协议规则 t 的卫士命令 $guard$, 我们可以计算出辅助不变量 $aux = ((g \& guard) => precond(f, t))$. 因为 aux 是协议不变量, 那么协议公式 f 在转移规则 t 执行后也成立. 因此对于转移规则集合中的每条规则 t, 协议状态公式 f 在规则 t 执行后都成立, 从而在可达状态集合中的任意状态 S 下都成立.

在缓存一致性协议的验证过程中, 关键是查找缓存协议的辅助不变量. 为了找到所有这些辅助不变量 g 来完成证明过程, 我们设计实现了一种协议不变量查找算法. 该方法从单个协议公式出发, 查找出其他辅助不变量. 在查找协议不变量的过程中, 我们将得到的协议不变量取反以方便计算, 即 $! inv = ! (u => v) = u \& (! v)$.

假设缓存一致性协议实例为 $P = (I, T)$, 协议初始状态为 I , 协议转移规则集合为 T. 协议系统中表示安全性质的公式为 f. 最后得到的辅助不变量集合为 M. 算法从公式 f 出发, 通过查找协议公式和转移规则之间的关系来计算出新的辅助不变量 aux.

算法开始时, 初始协议不变量集合 M 只有一个公式 f. 初始公式 f 一般是缓存一致性协议的安全性质, 比如控制性质: 不存在两个独占状态的缓存 $!(cache[i] = exclusive \& cache = exclusive)$. 实例化协议规则集合 T 后, 检查每条带参规则 t 和公式 f 之间存在的关系: 如果规则和协议公式之间存在关系 *InvHoldForRule1* 或者存在关系 *InvHoldForRule2*, 记录协议公式和协议规则之间的关系; 如果规则与协议

公式之间存在关系 *InvHoldForRule3*, 必须计算新的辅助不变量 *aux* 并将它加入不变量集合 *M* 同时记录公式和协议规则之间的关系, 循环这一过程直到不能推出新的辅助不变量, 查找过程结束, 返回不变量集合 *M* 和关系集合 *R*. 不变量查找过程如下所示.

```
Input: initial formula f, protocol instance P = (I, T)
Output: invariants set M, relations set R
Process:
formula queue Q := {f}
while true do
    f := Pop(Q)
    N := empty
    rs := Instantiate(T)
    while rs not empy do
        r := Pop(rs)
        f := PreCond(f, r)
        if invHoldForRule1(f, r) then
            InsertRel(R, rel1)
        elif invHoldForRule2(f, r) then
            InsertRel(R, rel2)
        else
            InsertRel(R, rel3)
            inv = Choose(f, r)
            Push(N, inv)
        end
    end
    if N is empty then
        return (M, R)
    else
        push(Q, N)
        push(M, N)
    end
end
```

图 1 不变量查找过程

如果规则与协议公式之间存在关系 *InvHoldForRule3*, 必须通过当前协议公式 *f* 和协议规则 *r* 来计算新的辅助不变量 *aux*. 首先从协议规则 *r* 构造出前提公式 *g*, 如果前提公式 *g* 和协议公式 *f* 组成的新逻辑公式不是缓存协议的不变量, 继续计算过程, 否则, 检查新不变量是否为永真式或者已经包含在不变量集合 *M* 中, 若否, 尝试简化得到的不变量, 最后返回新不变量. 这一过程如下所示.

```
Input: original formula f, rule r = (guard, Statement)
Ouput: new auxiliary invariant aux
Process:
    while true do
        g := select(r)
        pre := g & guard
        aux := (pre => !f)
        if aux not invariant then
            continue
        elif aux = True or aux in M then
            continue
        else
            aux := (pre & f)
            aux := refine(aux)
            return aux
        end
    end
```

图 2 计算辅助不变量过程

当不变量查找工具找出协议中所有必要的辅助不变量集合 *invs* 并得到每个不变量和转移规则对应的关系集合 *rels* 之后, 利用 *invs* 和 *rels* 自动产生协议的证明脚本文件. 在证明脚本中, 查找得到的辅助不变量转化为对应的带参形式, 通过之前证明的一致性引理构造带参证明. 最后, 我们将 German 协议的带参证明脚本放在网上[15], 可以利用定理证明工具如 Isabelle 和生成的证明脚本来检查协议正确性.

4.2 辅助定理证明

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[16]. Isabelle 为证明系统开发提供了一个通用框架. 因此, 我们使用 Isabelle 工具来帮助说明 German 缓存协议的正确性.

在查找不变量的同时, 程序也计算出不变量与规则的关系集合 *R*. 在关系集合 *R* 中, 对于不变量集合的每个不变量 *inv* 和 German 协议实例中的每条协议规则 *r*, 都存在关系 *InvHoldFor1~invHoldForRule3* 之间的一种. 根据不变量和协议规则对应关系以及定理 1,

我们使用 Isabelle 定理证明器构建 German 协议的形式化带参证明, 进而说明数据安全性质和控制安全性质在 German 缓存协议的每个状态下都是成立的. Isabelle 证明脚本包括控制信号, 规则, 不变量, 初始化公式, 定理及证明. 通过对这种寻找不变量的过程进行分析, 我们能够进一步理解 German 缓存协议设计的正确性.

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 German 协议进行了实验. 我们分别以数据安全性质(data property)和控制安全性质(control property)为初始公式对 German 协议进行不变量查找, 然后利用实验得到的辅助不变量集合和对应关系集合生成证明脚本并使用 Isabelle 定理证明工具验证 German 协议的正确性. 下面是分别对协议的控制部分性质和数据部分性质进行实验得到的实验结果. 从实验结果可以看出, German 协议的控制性质可以在较短时间内完成, 对于复杂的数据性质用时较长, 实验产生的辅助不变量数目比控制性质多.

表 1 German 协议实验结果

	#Rule	#Inv	Time(s)	Mem(Mb)
控制性质	13	24	21.09	26.7
数据性质	15	50	45.71	29.4

同时实验得到的辅助不变量可以用来分析和验证 German 协议的正确性. 事实上, 这些不变量给出了协议性质的完整描述. 如协议保证不会存在两个(或以上)缓存状态同时为互斥状态: (!(*cache[i1].state=exclusive&cache[i2].state=exclusive*)), 当缓存状态为互斥状态时对应的 *shrSet* 位应该置为 True: (!(*cache[i1].state=exclusive & shrSet[i1]=false*)) 等. 辅助不变量所反映的协议性质也是对协议运行过程的说明, 如表 2 所示

表 2 部分不变量的协议说明

Invariant(negative form)	Protocol Meaning
(cache.state[1]=exclusive)&(cache.state[2]=exclusive)	不存在两个缓存状态同时为互斥状态
(cache.state[1]=exclusive)&(channel2_4.cmd[2]=grant_exclusive)	当节点 1 的缓存状态为互斥时, Home 不会向节点 2 发送同意互斥消息
(channel2_4.cmd[1]=grant_exclusive)&(channel2_4.cmd[2]=grant_exclusive)	Home 不会同时发送的同意互斥消息到节点 1 和节点 2
(cache.state[1]=exclusive)&(home_sharer_list[1]=false)	当节点 1 状态为互斥时, 对应的 home_sharer_list 应该置为 true
(channel2_4.cmd[1]=grant_exclusive)&(home_sharer_list[1]=false)	Home 向节点 1 发送同意互斥消息后, 对应的 home_sharer_lsit 置为 true
(cache.state[1]=exclusive)&(channel3.cmd[1]=invalidate_ack)	当节点 1 的状态为互斥时, 不会向 Home 发送无效应答
(channel2_4.cmd[1]=grant_exclusive)&(channel3.cmd[1]=invalidate_ack)	Home 向节点 1 发送同意互斥消息后, 节点 1 不会发送无效应答
(channel3.cmd[1]=invalidate_ack)&(home_sharer_list[1]=false)	在节点 1 向 Home 发送无效应答时, 这时的 home_sharer_list 为 true
(home_sharer_list[1]=false)&(channel2_4.cmd[1]=invalidate)	如果节点 1 的 home_sharer_list 为 false, Home 不会向它发送无效消息
(channel2_4.cmd[1]=invalidate)&(channel3.cmd[1]=invalidate_ack)	Home 向节点 1 发送无效消息和节点 1 的无效应答不会同时发生

最后, 我们利用实验得到的辅助不变量集合和对应关系集合生成证明脚本并使用 Isabelle 定理证明工具验证 German 协议的正确性. 完整的不变量集合和最终生成的证明脚本我们已经放在网站[15]上.

6 结论

German 缓存一致性协议是带参验证中经常使用的一种带参协议. 我们提出了通过查找缓存一致性协议不变量来验证带参协议正确性的新方法. 这种方法通过寻找不变量和协议规则之间的对应关系来计算辅助不变量并利用定理证明工具辅助证明, 从而帮助推导验证缓存一致性协议. 最后, 我们将方法应用到 German 缓存协议上计算它的辅助不变量来验证它的数据性质和控制性质. 我们通过查找辅助不变量来分析和验证协议设计的正确性. 这些不变量也是协议性质的直观描述, 从而帮助我们更好地理解带参协议的正确性.

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 孙鲁民,周琰.RCC 高速缓存一致性协议的带参验证.计算机系统应用,2014,23(11):10–15.

5 Park S, Dill DL. Protocol verification by aggregation of distributed Trans. Computer Aided Verification, 1996: 300–310.

6 Clarke E, Long D, McMillan K. Compositional model checking. LICS, 1989: 353–362.

7 Cadence Berkeley Labs. Cadence smv, http://www.kenmcmil.com/smv.html. 1998.

8 McMillan KL. Parameterized verification of the FLASH cache coherence protocol by compositional model checking. CHARME, 2001: 179–195.

9 McMillan KL. Verification of infinite state systems by compositional model checking. CHARME, 1999: 219–237.

10 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.

11 Conchon S, Goelz A, Krsticz S. Invariants for finite instances and beyond. Formal Methods in Computer-Aided Design, 2013: 61–68.

12 Conchon S, Goel A, Krstic S, Mebsout A, Zaidi F. Cubicle: a parallel SMT-based model checker for parameterized systems. Computer Aided Verification, 2012: 718–724.

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

14 Sorin DJ, Hill MD, Wood DA. A primer on memory consistency and cache coherence. Synthesis Lectures on Computer Architecture, 2011, 6(3): 3–6.

15 German Protocol Proof Script, http://lcs.ios.ac.cn/~lyj238/invFinder.html. 2015.

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