计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2017,26(10):156−160 [doi: 10.15888/j.cnki.csa.006020]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

基于流分析与归纳不变式结合的 German 协议验证①

张　瑜, 孙文辉

(北京交通大学 计算机与信息技术学院, 北京 100044)

摘　要: German 缓存一致性协议是用于共享内存的并发多处理器系统中的缓存一致性协议, 对 German 协议进行形式化验证一直是学术界和工业界的热点. 我们生成 German 协议的流图, 对流程图的各个步骤进行详细的描述, 并提出了流分析与归纳不变式结合对协议验证的方法, 通过辅助不变式与协议流图的对应关系, 从而进一步分析和验证 German 协议的正确性.

关键词: 缓存一致性协议; 流分析; 归纳不变式; 形式化验证

引用格式: 张瑜,孙文辉.基于流分析与归纳不变式结合的 German 协议验证.计算机系统应用,2017,26(10):156–160. http://www.c-s-a.org.cn/1003-3254/6020.html

Verification of German Cache Coherence Protocol by Flow Analysis and Inductive Invariants

ZHANG Yu, SUN Wen-Hui

(School of Computer and Information Technology, Beijing Jiaotong University, Beijing 100044, China)

Abstract: German cache coherence protocol is used in parallel multi-processor systems, and the verification of German protocol has always been a hot spot in international industry and academia. We generate the flow chart of German protocol and describe each step of the flow chart. Besides, we present a method to verify the cache coherence protocol by flow analysis and inductive invariants in this paper. By searching for the relations between the invariants and the flow chart of German protocol, we can further analyze and verify the correctness of German protocol.

Key words: cache coherence protocol; flow analysis; inductive invariants; formal verification

1　引言

带参系统广泛存在于计算机体系的核心模块中, 通常由参数个具有相同结构的并发执行的主体和有限个结构不同的主体组成. 带参系统在很多领域都有实际的应用, 如缓存一致性协议, 安全协议, 网络通信协议等, 都可以用带参系统描述.

German 缓存一致性协议是作为形式化验证领域的一个挑战性问题于 2000 年由 Steven German 提出的一个基于地址目录的缓存一致性协议, 是带参验证中广泛使用的例子. 常用的验证策略有两种, 即基于模型检测的技术[1]和基于定理证明的技术[2]. 目前, 有很多人采用不同的方法对 German 协议进行了形式化验证, 如: 吕毅等采用了参数抽象与卫士加强[3]的方法; Baukus 等采用谓词抽象[4]的方法; Alan hu 采用截止的方法[5]; 曹燊等用了不变式查找[6]的方式, 等等. 但是这些方法中对 German 协议内容的描述都是不完整的, 只有部分文字性的说明, 这对于工程师而言不够直观和明确, 也不易于准确理解协议的设计. 针对这一问题, 我们在 German 协议的验证中引入了流图的概念, 协议流图是对协议内容的一个图形描述, 在逻辑上精确地描述了协议的功能, 以图形的方式描述消息在协议流程中流动和处理的迁移过程, 可以有效地帮助用户理解、分析协议的设计.

本文通过生成 German 协议的流图, 并将流图分析

① 基金项目: 国家自然科学基金 (61672503)
收稿时间: 2017-01-12; 采用时间: 2017-02-23

与归纳不变式结合起来对 German 协议进行验证. 本文的工作主要体现在以下方面:

① 生成 German 协议规则的完整的流图, 帮助人们准确理解 German 协议的设计;

② 将生成的辅助不变式与 German 协议的规则结合起来对不变式的含义进行说明;

③ 通过将辅助不变式与 German 协议的流图对应结合, 根据 German 协议的流图解释这些辅助不变式的含义, 进一步对协议的设计进行说明.

2 German 协议描述

2.1 协议内容描述

German 协议是 Steven german 2000 年提出的基于地址目录的缓存一致性协议[7], German 协议主要适用共享存储的并发多处理器系统, 用来维护每个节点缓存的一致性. 在这个协议中, 有一个维护目录的主节点 Home 和 N 个同构的用户节点 Client, Home 节点作为地址目录的一个中心控制部分, 并拥有 memory, 地址目录的功能是记录各个 Client 中的 Cache 状态 (无效 I、共享 S、独占 E), 多个 Client 节点 (从控制中心申请共享或独占一个内存地址). cache 通过请求共享或者独占 memory, 将 memory 中的数据读入 cache.

Home 和 Clients 节点之间的消息分为 4 种, 如图 1 所示.

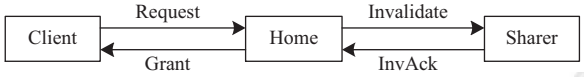

图 1 Home 和 Client 节点传递的消息

从图 1 可以看出这四种消息分别为:

① Client 向 Home 节点发送的请求共享或者独占内存的请求消息;

② Home 向一个共享或独占内存的 Client 节点发送的无效请求消息;

③ 共享或独占内存的 Client 节点发送的无效响应消息;

④ Home 向有请求的 Client 节点发送的请求响应消息.

Home 和 Clients 节点之间的消息是通过三个单向的消息通道传递的, 如图 2 所示.

从图 2 可以看出这三个单向的消息通道分别是:

① chan1 处理 Client 向 Home 发送的请求消息 (ReqS、ReqE);

② chan2 处理 Home 向 Client 发送的无效消息 (Inv) 和请求响应的消息 (GntS、GntE);

③ chan3 处理 Client 向 Home 发送的无效响应消息 (InvAck).

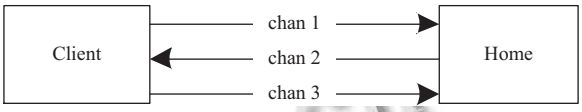

图 2 Home 和 Client 之间的消息通道

2.2 协议流图描述

协议流图对协议的形式化验证是有促进作用的. 首先, 流图描述使协议中各个节点之间执行规则的因果关系变得非常清晰, 也对各个节点之间的通信过程进行了说明, 易于协议设计的理解; 其次, 流图分析直观的表示了协议中消息的迁移过程, 有助于对协议进行研究与验证.

我们详细分析了 German 协议的内容, 并生成了 German 协议的规则流程图, 对流程图中的步骤做了具体的说明, 如图 3、4 所示.

图 3 是 Client 节点向 Home 节点发送的请求共享的消息, 具体步骤如下.

① 一个 Client 节点请求需要共享数据缓存副本, 执行规则 SendReqS, 向 Home 发送 ReqS 请求;

② Home 节点接收到 ReqS 请求后查询目录 directory 的状态信息, 执行规则 RecvReqS;

③ 当没有其他 Client 节点处于独占状态时, 则直接从 memory 中读取数据, 执行规则 SendGntS, Home 节点向有请求的 Client 节点发送 GntS 消息;

④ 当有其他 Client 节点处于独占状态时, 执行规则 SendInv, Home 节点先对这些独占状态的 Client 节点发送 Invalidate 消息, 使独占缓存无效; 独占状态的 Client 节点发送无效响应消息, 执行规则 SendInvAck, 向 Home 节点发送 InvAck 消息, Home 节点收到 InvAck 消息, 执行规则 RecvInvAck, 从 InvAck 消息中读取数据, 执行规则 SendGntS, Home 节点向有请求的 Client 节点发送 GntS 消息;

⑤ 有请求的这个 Client 节点收到 GntS 消息, 执行规则 RecvGntS, 将缓存副本的状态为共享.

图 4 是 Client 节点向 Home 节点发送的请求独占

内存的消息, 具体步骤如下.

① 一个 Client 节点请求需要独占数据缓存副本, 执行规则 SendReqE, 向 Home 发送 ReqE 请求;

② Home 节点接收到 ReqE 请求后查询目录 directory 的状态信息, 执行规则 RecvReqE;

③ 当没有其他 Client 节点处于独占状态时, 则直接从 memory 中读取数据, 执行规则 SendGntE, Home 节点向有请求的 Client 节点发送 GntE 消息;

④ 当有其他 Client 节点处于独占状态时, 执行规则 SendInv, Home 节点先对这些独占状态的 Client 节点发送 Invalidate 消息, 使独占缓存无效; 独占状态的 Client 节点发送无效响应消息, 执行规则 SendInvAck, 向 Home 节点发送 InvAck 消息, Home 节点收到 InvAck 消息, 执行规则 RecvInvAck, 从 InvAck 消息中读取数据, 执行规则 SendGntE, Home 节点向有请求的 Client 节点发送 GntE 消息;

⑤ 有请求的这个 Client 节点收到 GntE 消息, 执行规则 RecvGntE, 将缓存副本的状态为独占, 并执行规则 Store, 将数据写入缓存副本中.

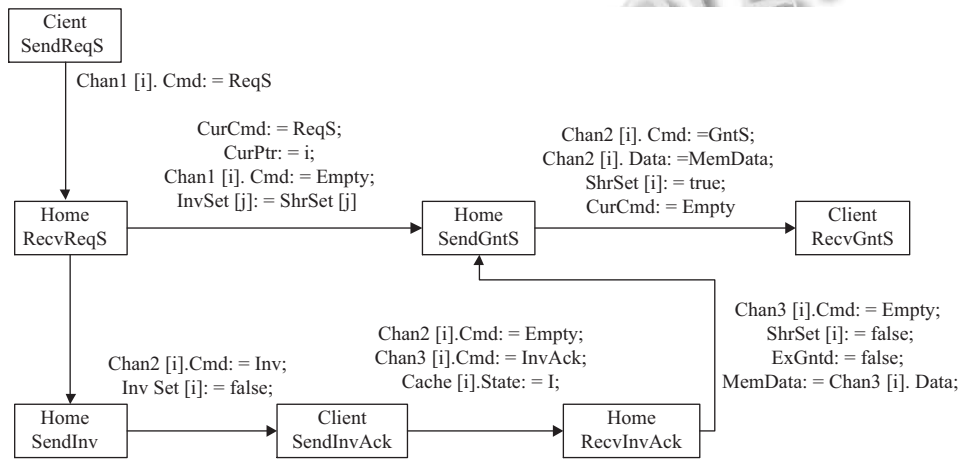

图 3 German 协议 ReqS 请求

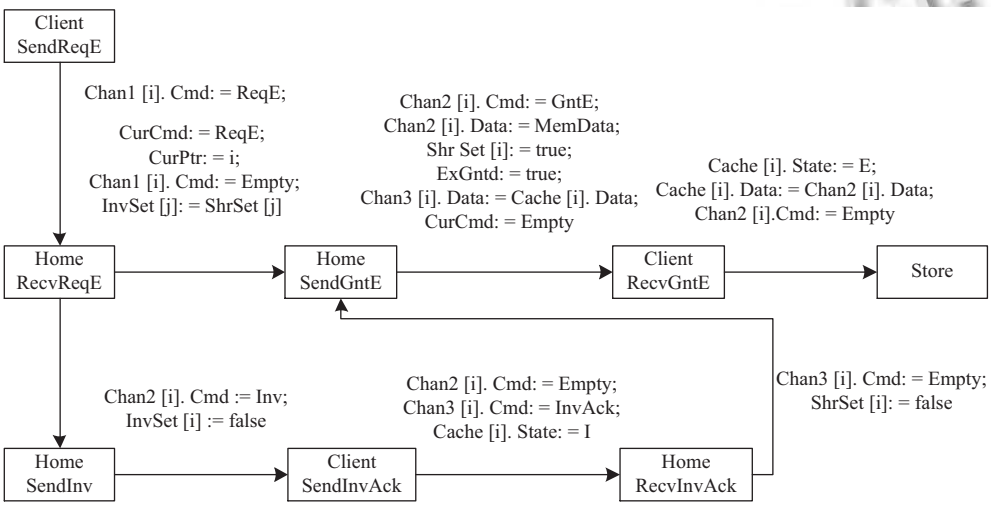

图 4 German 协议 ReqE 请求

3 German 协议的归纳不变式

3.1 辅助不变式

我们在四核 Intel Xeon 2.4 GHz 处理器, 8 GB 内存, 64 位 Linux 3.15.10, Murphi 版本为 cmurphi5.4.9 的环境下对 German 协议的 Murphi[8,9]模型进行了实验, 实验结果如表 1 所示.

表 1 German 协议的实验结果

Rules	12
Invariants	69
Time (s)	68.355
Memory (MB)	62.8

从实验结果可以看出, German 协议共有 69 个辅助不变式, 我们已经把查找到的所有辅助不变式放到了网上[10]. 我们选择了有关 ExGntd 的所有不变式进行具体的描述分析, 这些不变式也是协议性质的直观描述, 帮助我们理解协议变量的功能.

```
inv__21: ((ExGntd = FALSE) & ((MemData =AuxData)))
inv__27: ((ExGntd = FALSE) & (!Cache[1].State = e))
inv__28: ((!(Chan3[1].Data = AuxData)) & (ExGntd = TRUE) &
(Chan3[1].Cmd = invack))
inv__36: ((ExGntd = FALSE) & (Chan2[1].Cmd = gnte))
inv__37: ((ExGntd = TRUE) & (!(Cache[1].State = e))
& (Chan2[1].Cmd = inv))
inv__45: ((ExGntd = TRUE) & (!(Cache[1].State =e))
& (Chan2[1].Cmd = empty) &(InvSet[1] =TRUE))
inv__50: ((ExGntd = TRUE) & (!(Cache[1].State = e))
& (Chan2[1].Cmd = empty) &(ShrSet[1] =TRUE) & (CurCmd = empty))
inv__53: ((ExGntd = TRUE) & (Chan2[1].Cmd =gnts))
inv__55: ((Chan3[1].Cmd = invack) & (CurCmd =reqs) & (ExGntd = FALSE))
inv__61: ((CurCmd = reqs) & (ExGntd =FALSE) & (Chan2[1].Cmd = inv))
inv__68: ((InvSet[1] = TRUE) & (ExGntd =TRUE) & (InvSet[2] = TRUE))
inv__69: ((ShrSet[2] = TRUE) & (ExGntd =TRUE) & (ShrSet[1] = TRUE))
```

图 5 部分不变式

inv_21: (ExGntd=FALSE) Client 节点的独占状态标识为 FALSE, 那么内存中的数据就一定是正确的数据.

inv_27: 节点 1 的缓存状态是独占状态, 则该节点的独占状态标识为 TRUE.

inv_28: 节点 1 发送无效应答给 Home 节点, 而节点的独占状态标识为 TRUE, 则节点 1 发送的数据是正确的数据.

inv_36: Home 节点向节点 1 发送同意独占消息, 则该节点的独占状态标识为 TRUE.

inv_37: 节点的独占状态标识为 TRUE, 并且节点 1 的 cache 状态不是独占状态, 则 Home 节点不会向节点 1 发送无效消息.

inv_45: 节点 1 的缓存状态不是独占状态, Home 发送空消息给节点 1, 节点 1 对应的 InvSet 为 TRUE, 则独占状态标识为 FALSE.

inv_50: 节点 1 的缓存状态不是独占状态, Home 发送空消息给节点 1, 节点 1 对应的 ShrSet 为 TRUE, Home 当前命令为空, 则独占状态标识为 FALSE.

inv_53: Home 节点向节点 1 发送同意共享消息, 则该节点的独占状态标识为 FALSE.

inv_55: 节点 1 发送无效应答给 Home 节点, 而节点的独占状态标识为 FALSE, 则 Home 当前命令是请求共享.

inv_61: Home 向节点 1 发送无效消息, 而节点的独占状态标识为 FALSE, 则 Home 当前命令是请求共享.

inv_68: 节点 1 对应的 InvSet 为 TRUE, 且独占状态标识为 TRUE, 则节点 2 对应的 InvSet 为 FALSE.

inv_69: 节点 2 对应的 ShrSet 为 TRUE, 且独占状态标识为 TRUE, 则节点 1 对应的 ShrSet 为 FALSE.

3.2 转移规则与辅助不变式

我们选取规则 RecvGntS 和不变式 ((Cache[1].State = e) & (!(Cache[2].State = i))) 进行具体的说明.

```
ruleset i : NODE do rule "RecvGntS"
  Chan2[i].Cmd = GntS ==> begin
  Cache[i].State := S;
  Cache[i].Data := Chan2[i].Data;
  Chan2[i].Cmd := Empty; endrule; endruleset;

1 rule: n_RecvGntS[1]; inv: ((Cache[1].State = e) & (!(Cache[2].State = i)));
g: TRUE; rel: invHoldForRule1
2 rule: n_RecvGntS[2]; inv: ((Cache[1].State = e) & (!(Cache[2].State = i)));
g: TRUE; rel: invHoldForRule3 -inv__22:((Cache[1].State = e)
& (Chan2[2].Cmd = gnts))
3 rule: n_RecvGntS[3]; inv: ((Cache[1].State = e) & (!(Cache[2].State = i)));
g: TRUE;rel: invHoldForRule2
```

图 6 规则与不变式

转移规则 RecvGntS 的赋值部分不改变不变式 ((Cache[1].State = e) & (!(Cache[2].State = i))) 中的变量, 那么不变式 ((Cache[1].State = e) & (!(Cache[2].State = i))) 在规则 RecvGntS 执行后的状态 s_1 下成立.

如果状态 s 满足规则 RecvGntS 的卫士条件, 并且存在辅助不变式 inv__22:((Cache[1].State = e) & (Chan2[2].Cmd = gnts)), 执行规则 RecvGntS 后的状态是 s_1, 那么不变式 ((Cache[1].State = e) & (!(Cache[2].State = i))) 在状态 s_1 成立.

4 通过不变式描述一个典型流

辅助不变式可以用来分析和验证 German 协议的正确性, 给出协议性质的完整描述. 辅助不变式所反映的协议性质也是对协议运行过程的说明.

我们选取图 7 的部分不变式, 将这些不变式与 German

协议的 ReqS 请求的一个典型流结合, 对 German 协议的内容做更深层的分析, 帮助我们进一步理解 German 协议的设计.

```
inv__22: ((Cache[2].State = e) & (Chan2[1].Cmd = gnts))
inv__25: ((!(Chan2[1].Data = AuxData)) & (Chan2[1].Cmd = gnts))
inv__32: ((Chan2[1].Cmd = gnts) & (Cache[1].State = e))
inv__39: ((Chan2[1].Cmd = gnts) & (ShrSet[1] = FALSE))
inv__47: ((Chan2[1].Cmd = gnts) & (Chan3[1].Cmd = invack))
inv__53: ((ExGntd = TRUE) & (Chan2[1].Cmd = gnts))
```

图 7 部分不变式

当节点 1 请求共享, 没有其他节点处于独占状态时, 执行规则 SendGntS 时, Home 节点发送同意共享的消息到节点 1, 那么 Home 发送的数据一定是正确的数据, 节点 1 和节点 2 的缓存状态肯定不是独占状态, 节点 1 对应的 ShrSet 为 TRUE, 对应不变式 inv_22, 25, 32, 39 所示.

有其他节点处于独占状态, 执行规则 SendInvAck, 节点 1 发送无效应答消息给 Home, Home 肯定不会给节点 1 发送同意共享消息, 对应不变式 inv_47. 然后执行规则 SendGntS 时, Home 节点发送同意共享的消息到节点 1, 那么 ExGntd 一定为 FALSE.

5 结论

German 缓存一致性协议是带参验证中经常使用一种带参协议. 我们提出了流分析与归纳不变式结合对协议验证的方法, 这种方法实现了对 German 协议的流图分析, 查找到 German 协议的所有辅助不变式, 对这些辅助不变式的含义做了具体说明, 并将辅助不变式与 German 协议的流图对应结合, 进一步分析和验证了 German 协议设计的正确性. 这种方法相对于现有的方法, 对协议内容的描述更加直观和明确, 以图形和文字结合的方式对 German 协议的内容做了完整的描述, 流图分析在逻辑上精确描述了协议的功能和消息在协议流程中的迁移过程, 这对于人们理解和分析协议的内容和设计是非常有效的; 其次, 根据已查找到的辅助不变式描述 German 协议流图中的典型流, 这些辅助不变式是协议性质的直观描述, 将不变式与协议流图中的典型流结合, 是对协议的内容和迁移过程更直观的分析, 从而验证 German 协议设计的正确性.

参考文献

1 Clarke EM, Grumberg O, Peled DA. Model Checking. Cambridge: The MIT Press, 1999.

2 Nipkow T, Paulson LC, Wenzel M. Isabelle/HOL: A proof assistant for higher-order logic. Berlin Heidelberg: Springer, 2002.

3 Lv Y, Lin HM, Pan H. Computing invariants for parameter abstraction. Proc. of the 5th IEEE/ACM International Conference on Formal Methods and Models for Codesign. Nice, France. 2007. 29–38.

4 Baukus K, Lakhnech Y, Stahl K. Parameterized verification of a cache coherence protocol: Safety and liveness. Proc. of the 3rd International Workshop on Verification, Model Checking, and Abstract Interpretation. Venice, Italy. 2002. 317–330.

5 Bingham J, Hu AJ. Empirically efficient verification for a class of infinite-state systems. Proc. of the 11th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. Berlin Heidelberg, Germany. 2005. 77–92.

6 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): 173–178. [doi: 10.3969/j.issn.1003-3254.2015.11.028]

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

8 Dill DL. The Murphi verification system. Proc. of the 8th International Conference on Computer Aided Verification. Berlin Heidelberg, Germany. 1996. 390–393.

9 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. [doi: 10.3969/j.issn.1003-3254.2013.10.024]

10 All auxiliary invariants of German protocol. https://github.com/zy311/German/blob/master/invariants. [2017].