文章编号: 1001- 2486(2008) 04- 0053- 06

一种高效的显式模型检验方法*

屈婉霞, 李 暾, 郭 阳, 杨晓东
(国防科技大学 计算机学院, 湖南 长沙 410073)

摘 要: 随着软硬件系统规模和功能的不断扩充, 状态空间爆炸问题严重影响了模型检验的进一步发展与应用, 成为验证大规模系统的瓶颈。在显式模型检验工具 Murphi 的基础上, 针对其可达状态空间组织存在的问题进行改进, 提出了基于整型指针与 Fibonacci 散列的可达状态空间组织方法, 实现了一个高效的显式模型检验原型系统, 在确保验证正确的同时有效缩短了验证时间, 并能在系统规范不可满足的情况下给出反例, 有助于系统设计人员快速定位错误。理论分析和实验结果表明了本方法的有效性。

关键词: 模型检验; 显式状态枚举; 可达性分析
中图分类号: TP391. 72 文献标识码: A

An Efficient Explicit Model Checker

QU Wan-xia, LI Tun, GUO Yang, YANG Xiao-dong
(College of Computer, National Univ. of Defense Technology, Changsha 410073, China)

Abstract: With the growing increase in software/hardware system scale and function, the further development and application of model checking has been greatly limited by state space explosion, which has been the bottleneck of verifying large industrial designs. Based on the Murphi, an explicit state model checking tool, the problem of organization of reachable state space of model checking was studied, and a novel organization method of reachable state space based on integer pointer and Fibonacci hash was presented. In the approach, an efficient model checking system was suggested. The new approach can effectively shorten verification cycle, and give counter example when the system specification is unsatisfiable. All this greatly helped rapid error location. The analysis and experiment results prove the effectiveness of our method.

Key words: model checking; explicit state enumeration; reachability analysis

随着软硬件系统规模和功能的不断扩充, 系统的正确性验证日益困难。以模型检验为代表的形式化验证方法受到越来越多的关注, 基本思想是对系统模型和系统规范进行形式化描述, 然后通过遍历状态空间检验系统规范在系统模型中是否成立。在遍历过程中, 有两种描述可达状态空间的方法: 一种是用表格等结构显式地保存所有可达状态, 称为显式枚举, 如 SPIN[1]、Murphi[2] 等; 另一种是采用符号方法隐含地表示可达状态空间, 如基于二叉决策图的 SMV[3] 等。模型检验的发展一直在克服内存爆炸这一瓶颈带来的问题, 尽管符号方法能够有效缓解状态空间爆炸问题, 所能处理的状态数目达到 10^{20} 以上, 但是在某些情况下, 显式枚举方法使用更方便, 并且性能优于符号方法[4]。目前, 显式状态模型检验已在 Cache 一致性协议、存储模型、互斥协议、通讯协议等领域取得很好的应用效果[5-6]。

近年来, 显式模型检验采用了很多新技术以处理更复杂的系统。这些技术可分为两类。第一, 采用对称化简或偏序等状态化简方法[7], 确保在能够检测出系统错误的情况下减小可达图的规模。第二, 用更高效的方法搜索可达图, 减少内存使用和运行时间。例如 hash 压缩和状态空间搜索并行化[8] 等。

运行时间和内存仍然是大规模系统显式状态模型检验的主要瓶颈。本文在深入分析 Murphi 工作原理的基础上, 针对其可达状态空间组织所存在的问题进行改进, 提出了基于整型指针与 Fibonacci 散列的可达状态空间组织方法, 实现了一个高效的显式状态模型检验系统, 在确保验证正确的同时有效缩

* 收稿日期: 2007- 12- 13
基金项目: 国家自然科学基金资助项目(60573173, 60773025); 新世纪优秀人才支持计划资助项目
作者简介: 屈婉霞(1972—), 女, 助理研究员, 博士生。

短了验证时间, 并能在系统规范不可满足的情况下给出反例, 有助于系统设计人员快速定位错误。

1 Murphi 系统及其可达状态空间组织

Murphi 是 Utah 大学开发的显式模型检验工具, 它由 Murphi 描述语言、Murphi 编译器和 C+ + 编译器组成。Murphi 描述包括常量、类型、全局变量和过程声明、迁移规则集合、初始状态集合和属性集合等。编译器将 Murphi 描述转换为 C+ + 描述, 再经过 C+ + 编译器生成专用 Murphi 验证器, 以深度优先或宽度优先方式搜索状态机: 在当前状态找到所有条件为真的状态迁移规则, 任意选择一条规则执行, 产生一个新状态, 并根据验证者提供的参数对其进行相应检查。

以宽度优先方式搜索状态机时, Murphi 验证器使用状态队列(State Queue) 和状态表(State Table) 两个数组显式地保存状态, 如图 1 所示。状态队列是先进先出队列, 保存所有等待被检测的活跃状态。状态队列中的项是指向状态的指针。状态队列容量与状态表容量之比为常量 gPercentActiveStates。状态表以 Cache 方式组织, 保存已访问的所有状态并用于检测活跃状态是否被访问过。状态表每个项包括 bits 和 previous 两个域, 其中, bits 是状态编码, previous 是指向当前状态父结点的指针。状态的关键字由 bits 确定。双重散列函数把关键字为 hash(bits) 的状态映射到状态表中的特定位置。

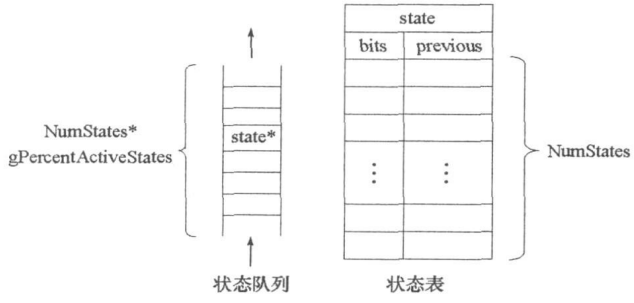

图 1 Murphi 可达状态空间组织
Fig. 1 Reachable space in Murphi

多个状态映射到状态表中同一个位置的情形称为碰撞(或冲突)。Murphi 采用双重散列和线性开型寻址方法处理散列过程中的碰撞, 将具有相同关键字的状态限制在长度为 MAX _HT _CHAIN _LENGTH 的表格区间内。当检测一个活跃状态是否被访问过时, 只需在相应表格区间内比较判断。当插入状态时, 如果状态表中已保存 MAX _HT _CHAIN _LENGTH 个与新状态冲突的状态, 系统将从相应表格区间内随机选择一个进行覆盖。

虽然双重散列能够缓解散列过程中的"聚集"现象, 但在出现冲突时需要计算另一个散列地址, 相应增加了计算时间。通过限制冲突次数加快状态表访问的方法, 每检测一个活跃状态, 需执行 4 次运算, 每插入一个状态, 需执行 3 次运算。如果关键字选择不合理或系统规模较大, 状态遍历过程中发生冲突和执行覆盖的次数就会比较多, 导致验证时间增加。更重要的是, 由于状态表的覆盖操作, previous 域不再指向其父结点, 当系统规范不满足时, Murphi 无法给出反例。此外, 很可能出现在状态表不满的情况下, 通过覆盖操作插入新状态项, 即状态表利用率小于 100%。

2 基于整型指针与 Fibonacci 散列的显式模型检验

2.1 基于整型指针的链表结构

针对上述问题, 提出了一种新的可达状态空间组织方法, 如图 2 所示。为状态表中的项增加整型指针域 next, 指向与当前状态冲突的另一状态项。关键字相同的所有状态项通过 next 链接, 称为冲突链。增加与状态表项数相同的入口表(Entry Table), 保存冲突链的第一个项在状态表中的位置。

这种基于整型指针的链表结构融合了线性开型寻址和指针链表处理碰撞的优点: 活跃状态的搜索和插入操作仅限于入口表指示的冲突链, 整型指针的使用加快了冲突链的访问速度; 入口表的项数不必与状态表项数完全相同, 可通过选择合适的散列函数优化冲突链的平均长度, 减少入口表的项数; 冲突

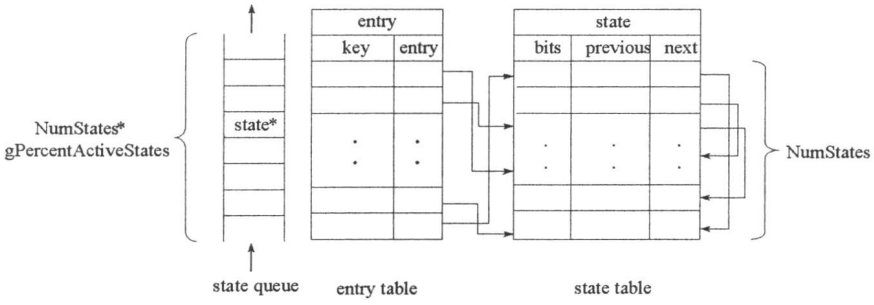

图 2 改进的可达状态空间组织

Fig. 2 Improved reachable space

链长度不受人为限制, 提高了状态表的利用率; 状态表操作的运算次数由原有的 4 次和 3 次分别减少到 2 次和 1 次, 缩短了系统验证时间; 只有在状态表有空闲项可用时, 新的活跃状态才可插入, 已访问状态不再被覆盖, 可在系统规范不满足时给出反例。

2.2 基于 Fibonacci 的散列优化

Fibonacci 散列法是一种特殊的相乘散列法, 与黄金分割 φ 密切相关。黄金分割的定义如下: 给定两个正数 x 和 y, 如果 x 与 y 的比值 x/y 等于 $x+y$ 与 x 的比值, 则称 $\varphi = x/y$ 为黄金分割。用这种方法散列连续关键字时, 每一个后续关键字的散列值落在已知的两个距离最大的散列值之间, 并对该区间进行黄金分割, 因此连续关键字分布非常均匀, 能够有效降低冲突率。

我们在新的数据结构上采用 Fibonacci 散列法, 优化冲突链的平均长度, 以期获得最优的平均访问性能。

2.3 算法描述

改进后的 Murphi 系统保留所有已访问状态, 不对状态表进行删除。下面给出在基于整型指针的链表结构中搜索和插入一个状态的过程描述以及专用验证器的工作过程描述。

1. *在状态表中检查关键字为 key 的状态 s 是否已被访问过*

Step 1 调用 Fibonacci 函数求 s 在入口表中的位置 pos;

Step 2 如果入口表的 pos 项为空, 表示首次访问到关键字为 key 的状态, 调用插入算法将 s 插入状态表并在入口表的 pos 项中记录插入位置, 返回 FALSE;

Step 3 如果入口表的 pos 项不空, 表示状态表中存在与 s 冲突的状态, 假设入口表指示 s 所在的冲突链在状态表中的入口为 h。

Step 3.1 如果状态表 h 项存储的状态与 s 相同, 表示状态 s 已被访问过, 返回 TRUE;

Step 3.2 如果状态表 h 项有后继, 转 Step 3.1, 否则调用插入算法将 s 插入状态表并将其链接到状态表 h 项的 $next$ 域, 返回 FALSE。

2. *在状态表中插入关键字为 key 的状态 s*

Step 1 如果状态表已满, 失败返回;

Step 2 状态表中元素的个数 num_elts 指示的位置即为第一个空闲项, 将插入状态表的第 num_elts 项, 成功返回。

3. *专用验证器的验证过程*

Step 1 产生所有初始状态, 并放入状态队列;

Step 2 如果状态队列为空, 打印验证报告并返回;

Step 3 从状态队列中取出头元素作为当前状态, 由规则集合求出所有下一状态, 并放入状态队列;

Step 4 若当前状态满足系统规范, 转 Step 2, 否则在状态表中沿 previous 域找到反例并返回。

3 算法分析

3.1 验证性能分析

设改进前后的Murphi系统均能在有限时间内完成验证工作,可达状态分布在 $ListNum$ 条冲突链中,第 i 条冲突链的长度为 L_i, $1 \leqslant i \leqslant ListNum$,如图3所示,状态项之间的指针实际不存在,仅表示状态项之间的连接关系, MAX 表示Murphi系统中的常量MAX_HT_CHAIN_LENGTH。

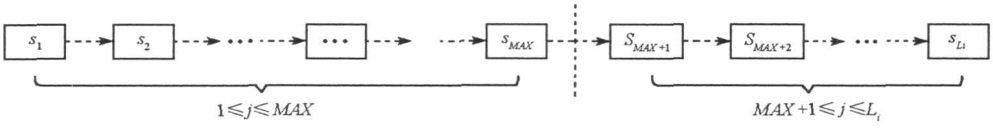

图3 第 i 条冲突链

Fig. 3 The ith collision list

为简化讨论,我们仅考虑构造第 i 条冲突链的运算开销,不考虑第 j 个状态($j \leqslant L_i$)被重复访问的情形,即每一个状态都是第一次被访问并插入到冲突链中。

令 A_φ 表示计算散列地址的操作, E_φ 表示判断状态表项是否为空的操作, C_φ 表示判断两个状态(或者数值)是否相同的操作, I_φ 表示插入一个状态的操作。在原Murphi系统中, A_φ 操作包括1次加法运算(用 C_+ 表示运算开销)、1次乘法运算(用 C_* 表示运算开销)和1次求余运算(用 $C_\%$ 表示运算开销), E_φ 操作和 C_φ 操作都只包括1次关系运算(用 $C_{==}$ 表示运算开销), I_φ 操作直接写内存,不执行任何运算。在改进后Murphi系统中, A_φ 操作包括1次加法运算、1次乘法运算和1次位运算, E_φ 操作和 C_φ 操作涉及的运算与原Murphi系统相同, I_φ 操作同样不执行任何运算。

在原Murphi系统中,冲突的 L_i 个状态"共享"状态表的MAX个表项,在一条冲突链中插入第 j 个状态和构造第 i 条冲突链的运算开销分别为:

$$c_j^p = \begin{cases} j*(c_+ + c_* + c_\%) + j*c_{==} + 2*(j-1)*c_{==} & 1 \leqslant j \leqslant MAX \\ (MAX+1)*(c_+ + c_* + c_\%) + MAX*c_{==} + 2*MAX*c_{==} & MAX+1 \leqslant j \leqslant L_i \end{cases} \tag{1}$$

$$C_i^p = \begin{cases} \dfrac{(c_+ + c_* + c_\% + 3*c_{==})*L_i^2 + (c_+ + c_* + c_\% - c_{==})*L_i}{2} & L_i \leqslant MAX,\ 1 \leqslant i \leqslant ListNum \\ [(c_+ + c_* + c_\% + 3*c_{==})*MAX + (c_+ + c_* + c_\%)](L_i - MAX) + \\ \dfrac{(c_+ + c_* + c_\% + 3*c_{==})*MAX^2 + (c_+ + c_* + c_\% - c_{==})*MAX}{2} & L_i > MAX,\ 1 \leqslant i \leqslant ListNum \end{cases} \tag{2}$$

在改进后的系统中,冲突链的长度不再受 MAX 约束,只要状态表中有空闲项可用,状态即可追加到冲突链中。在一条冲突链中插入第 j 个状态和构造第 i 条冲突链的运算开销分别为:

$$c_j^n = (c_+ + c_* + c_{>>}) + j*c_{==} + (j-1)*c_{==}, \quad 1 \leqslant j \leqslant L_i \tag{3}$$

$$c_i^n = c_{==}*L_i^2 + (c_+ + c_* + c_{>>})*L_i, \quad 1 \leqslant i \leqslant ListNum \tag{4}$$

上述公式显示,在原Murphi系统中,构造一条冲突链的运算开销是 L_i 和 MAX 的函数,而改进后系统构造冲突链的运算开销只与 L_i 相关,因此可以在保留可能出现的反例的同时,通过设计合适的散列函数控制冲突链的平均长度,实现验证性能的优化。

3.2 验证能力分析

设内存阈值为 M,原Murphi系统状态表项数为 n_1,每一项占用内存为 $sizeof(state)$,改进后状态表项数为 n_2,每一项占用内存为 $sizeof(state) + sizeof(next)$,其中, $sizeof(state)$ 表示1个状态项占用的字节数,该值与系统规模成正比,用 S 表示系统模型,则有 $sizeof(state) = k*|S|$,其中, k 为比例常数, $sizeof(next)$ 为 $next$ 域占用的内存字节数,大小为4字节。两个系统中状态队列的项均是指向状态的指针,大小为4字节,状态队列项数与状态表项数比值为gPercentActiveStates(简写为 gP)。

原 Murphi 系统有下列等式成立:

$$n_1 * gP * sizeof(state*) + n_1 * k * |S| = M \tag{5}$$

式中, 第一项是状态队列占用的内存, 第二项是状态表占用的内存。

改进后系统有下列等式成立:

$$n_2 * gP * sizeof(state*) + n_2 * (k * |S| + 4) + n_2 * 8 = M \tag{6}$$

式中, 第一项是状态队列占用的内存, 第二项是状态表占用的内存, 第三项是入口表占用的内存, 其项数与状态表项数相等, 每一项的 *key* 域和 *entry* 域均占用 4 字节内存, 则有

$$n_1 = \frac{M}{gP * sizeof(state*) + k * |S|}, \quad n_2 = \frac{M}{gP * sizeof(state*) + (k * |S| + 4) + 8} \tag{7}$$

用 4 代替 $sizeof(state*)$, 得到

$$\frac{n_1}{n_2} = \frac{4 * gP + k * |S| + 12}{4 * gP + k * |S|} = 1 + \frac{12}{4 * gP + k * |S|} \tag{8}$$

gP 是常量, 取值范围为[0.1, 1], 从式(8)可以看出, 随着系统模型规模的不断扩充, 当$(4 * gP + k * |S|) \gg 12$ 时, n_1/n_2 趋近于 1, 即改进前后系统可保存的状态表项数目相等。

综合上述分析, 在验证大规模系统时, 通过设计适当的散列函数, 基于整型指针和 Fibonacci 散列的 Murphi 系统性能优于原 Murphi 系统, 而内存占用与原系统相当。

4 实验与结果

在 Cygwin 环境下实现了基于整型指针与 Fibonacci 散列的快速 Murphi 系统, 并通过实验与原 Murphi 系统进行了分析比较。采用了 newcache3、arbiter、mcslock1、N_peterson 四个实验用例。newcache3 是基于目录的 Cache 一致性协议, 结点通过消息传递实现对共享内存的一致性访问。arbiter 是同步仲裁器协议, 部件之间竞争令牌 token 共享网络资源。mcslock1 协议采用链式队列锁在共享存储多处理机中实现了可扩展的同步算法, 保证多个进程对共享临界区的互斥访问。N_peterson 是在支持读-写原子操作的系统中实现多个进程对共享临界区互斥访问的经典协议, 参数 N 表示参与竞争的进程数目。这 4 个例子均用 Murphi 语言进行描述。

表 1 给出了改进前后的 Murphi 系统验证不同协议的结果对比。Murphi 编译器的参数为-cache, 表示以非压缩方式保存状态项, 可达状态空间占用的内存阈值为 10MB, MAX_HT_CHAIN_LENGTH 取值为 20。newcache3 的参数为 1 个 home 结点、4 个 processor 结点、1 个共享内存地址和 2 个数据值。arbiter 对 4 个部件进行仲裁。mcslock1 中参与竞争的进程数为 4。N_peterson 中参与竞争的进程数为 7。从表 2 可以看出, 改进后的 Murphi 系统保证了验证结果的正确性, 在系统规范不满足时给出了反例, 而原 Murphi 系统由于覆盖操作不能给出反例。

表 1 不同协议的验证结果对比

Tab. 1 Verification results of different protocols

协议名称	newcache3		arbiter		mcslock1		N_peterson	
	改进前	改进后	改进前	改进后	改进前	改进后	改进前	改进后
状态表项数	37 831	37 273	274 739	247 529	308 641	274 739	274 739	247 529
状态表空项数	3647	2492	273 751	246 426	285 060	251 094	159 396	84 231
覆盖次数	648	0	121	0	72	0	221304	0
覆盖率	0.02	0	0.11	0	0	0	0.66	0
验证时间(s)	9.48	9.92	0.10	0.10	1.34	1.44	41.12	29.23
验证结果	No error found	No error found	Invariant“no token lost” failed	Invariant“no token lost” failed	No error found	No error found	No error found	No error found
是否给出反例	-	-	否	是	-	-	-	-

表2是改进前后的Murphi系统验证不同规模的N_peterson协议的对比。N 表示参与竞争的进程数目,取值限定在2~10之间。M 表示可达状态空间的内存阈值(单位为MB)。原Murphi系统的覆盖率阈值为90%,当覆盖率大于此值时,验证过程自动停止。

表2 不同规模的协议验证结果对比

Tab.2 Verification results of protocol with different sizes

N		状态表项数	入口表空项数	覆盖次数	覆盖率	验证时间(s)	验证结果
2	前	609 757	–	0	0	0.12	No error found
(M=10)	后	490 201	490 177	0	0	0.17	No error found
6	前	308 641	–	2814	0.08	2.86	No error found
(M=10)	后	274 739	273 715	0	0	2.94	No error found
7	前	274 739	–	221 304	0.66	41.12	No error found
(M=10)	后	247 529	246 505	0	0	29.23	No error found
8	前	247 529	–	222 997	0.900 894	413.84	覆盖率超过阈值,停止验证!
(M=10)	后	225 227	224 203	0	0	62.97	状态表满,停止验证!
8	前	990 137	–	2 640 201	0.87	507.11	No error found
(M=40)	后	900 917	899 893	0	0	377	No error found

在 N 和 M 相同的情况下,改进前后的Murphi系统验证结果相同,即改进后系统的功能得到了保证。$N<7$ 时,两个系统的运行时间相当,$N \geqslant 7$ 时,改进后的Murphi系统验证时间明显减少,运行速度提高25%以上。此外,实验数据显示入口表的利用率不足1%,可以通过设计适当的散列函数减少入口表占用的内存。

5 结束语

对显式模型检验工具Murphi的可达状态空间组织所存在的问题进行了改进,提出了一种基于整型指针与Fibonacci散列的可达状态空间组织方法,实现了一个高效的显式模型检验原型系统,理论分析和实验结果表明:本文方法在确保验证正确的同时有效缩短了验证时间,并能在系统规范不可满足的情况下给出反例,有助于系统设计人员快速定位错误。

下一步,我们将研究谓词抽象和环境抽象等技术,进一步提高处理大规模复杂系统的能力,缓解状态爆炸的问题。

参考文献:

[1] Holzmann G J. The Model Checker SPIN[J]. IEEE Trans. on Software Engineering, 1997, 23(5):279-295.

[2] Dill D L, Drexler A J, Hu A, et al. Protocol Verification as a Hardware Design Aid[C]//IEEE International Conference on Computer Design, 1992: 522-525.

[3] The SMV Model Checking System[EB/OL]. http://www.cis.ksu.edu/santos/smv-doc/, 2003.

[4] Hu A J, York G, Dill D L. New Techniques for Efficient Verification with Implicitly Conjoined BDDs[C]//31st Design Automation Conference, 1994: 276-282.

[5] Chen X, Yang Y, Gopalakrishnan G, et al. Reducing Verification Complexity of a Multicore Coherence Protocol Using Assume/Guarantee[C]//Formal Methods in Computer Aided Design, 2006: 81-88.

[6] Chou C T, Mannava P K, Park S. A Simple Method for Parameterized Verification of Cache Coherence Protocols[C]//Formal Methods in Computer Aided Design, 2004: 382-398.

[7] Ip C N. State Reduction Methods for Automatic Formal Verification[D]. PhD Thesis, Standford University, 1996.

[8] Stern U, Dill D L. Parallelizing the Mur φ Verifier[C]//Computer Aided Verification, 9th International Conference, 1997: 256-267.