软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2018,29(6):1517−1526 [doi: 10.13328/j.cnki.jos.005461]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

互模拟准局部验证算法的扩展与实现*

郑晓琳[1], 邓玉欣[1], 付 辰[2], 雷国庆[1]

[1](上海市高可信计算重点实验室(华东师范大学),上海 200062)

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

通讯作者: 邓玉欣, E-mail: yxdeng@sei.ecnu.edu.cn

摘 要: 互模拟是并发系统分析和验证的一个重要概念.主要扩展了一种由Du和Deng提出的准局部算法,使其更加适用于一般的标记迁移系统.用Java实现扩展后的准局部算法与Fernandez和Mounier提出的局部算法.以VLTS为实验数据基准进行大量的实验,发现在大多数情况下,前者的性能比后者更好.同时,修改了算法使其能够验证模拟关系.最后,用Java实现对标记迁移系统进行转换,使算法同时可以验证弱互模拟关系.

关键词: 互模拟;标记迁移系统;扩展

中图法分类号: TP301

中文引用格式: 郑晓琳,邓玉欣,付辰,雷国庆.互模拟准局部验证算法的扩展与实现.软件学报,2018,29(6):1517−1526. http://www.jos.org.cn/1000-9825/5461.htm

英文引用格式: Zheng XL, Deng YX, Fu C, Lei GQ. Extension and implementation of the quasi-local algorithm for checking bisimilarity. Ruan Jian Xue Bao/Journal of Software, 2018,29(6):1517−1526 (in Chinese). http://www.jos.org.cn/1000-9825/5461.htm

Extension and Implementation of the Quasi-Local Algorithm for Checking Bisimilarity

ZHENG Xiao-Lin[1], DENG Yu-Xin[1], FU Chen[2], LEI Guo-Qing[1]

[1](Shanghai Key Laboratory of Trustworthy Computing (East China Normal University), Shanghai 200062, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

Abstract: Bisimilarity plays an important role in the analysis and verification of concurrent systems. In this paper, an optimization of the quasi-local algorithm of Du and Deng is proposed to make it applicable for general labeled transition systems. Both the optimized algorithm and the local algorithm of Fernandez and Mounier are implemented in Java, and experiment using the VLTS benchmark suite shows the former outperforms the latter in most cases. The algorithms are also modified to check similarity. Finally, a procedure for transforming labeled transition systems is implemented to facilitate checking weak bisimilarity.

Key words: bisimulation; labeled transition system; optimization; weak bisimulation

互模拟[1]是并发理论[2]中最重要的概念之一,它提供了一种有用的验证技术[3]来比较交互系统的可观察行为.如果它们能够一步步地匹配彼此的移动,则两个系统是互模拟的.检测互模拟的算法通常可以分为两类:全局算法和局部算法[4,5].

全局算法主要验证一个系统中任意两个状态是否互模拟(有时我们比较两个系统的初始状态,可事先把两个系统拼接成一个更大的系统),它需要预先生成一个整体的状态系统.相比较之下,局部算法只需要判定给定

* 基金项目: 国家自然科学基金(61672229, 61261130589); 上海市自然科学基金(16ZR1409100)

Foundation item: National Natural Science Foundation of China (61672229, 61261130589); Natural Science Foundation of Shanghai, China (16ZR1409100)

本文由形式化方法的理论基础专题特约编辑傅育熙教授、李国强副教授、田聪教授推荐.

收稿时间: 2017-06-28; 修改时间: 2017-09-01; 采用时间: 2017-11-06; jos 在线出版时间: 2017-12-28

CNKI 网络优先出版: 2017-12-29 13:19:07, http://kns.cnki.net/kcms/detail/11.2560.TP.20171229.1318.003.html

的两个状态是否互模拟,同时检查这两个状态可达到的后续状态的行为关系.Fernandez 和 Mounier[6]提出了第 1 种局部算法,他们的算法通过遍历的方式动态地验证两个状态是否互模拟,这个算法主要是一边检验等价关系一边增加要考察的状态.Du 和 Deng[7]提出了一种准局部算法来验证互模拟.对于一些类型的标记迁移系统(LTS),例如简单的 LTS 和确定型 LTS,准局部算法具有更好的时间复杂度.然而在许多实际运用中,我们遇到的不是简单的 LTS,因此原始的准局部算法并不适用.

本文提出了一种准局部算法的扩展算法,使其能够适用于一般的 LTS.我们在 Java 中实现了扩展算法,并将其与局部算法进行比较.我们以 VLTS 为基准[8],其中包含各种大小不同的 LTS,状态个数从几百到几十万.

通过实验,我们抽取了 20 多个具有代表性的例子进行验证,通过观察运行的时间来比较两种算法.在大多数的情况下,准局部算法确实比局部算法更快.特别是处理确定型的 LTS 时,准局部算法比局部算法有明显的优势.为了验证模拟关系,我们还修改了这两种算法.我们的实验也显示了类似的现象——在大多数情况下,准局部算法优于局部算法.

为了使实现的算法有更加广泛的应用,我们调查了生成 LTS 的方法,发现可以利用 Fiarce 语言通过 TINA, CADP 等工具生成 LTS,以及 CADP 提供的 LOTOS 语言转换为 LTS.随后可以用我们实现的算法进行验证.另外,我们利用弱互模拟的定义对 LTS 进行饱和处理,再用我们实现的算法验证,极大地丰富了算法的可适用性.

本文第 1 节给出一些基本的准备知识.第 2 节回顾典型的互模拟算法.第 3 节介绍扩展的准局部算法.第 4 节将扩展算法的时间效率与局部算法时间效率进行比较.第 5 节考虑弱互模拟的情况,并提取实例进行实验计算运行时间.第 6 节总结本文的贡献与不足,并阐明后续的一些工作方向.

1 准备知识

我们将回顾操作模型的一些基本定义以及互模拟[1,2]的概念.在计算机科学中,迁移系统常常被用作描述系统行为的操作模型,它们是一种有向图,其中,节点代表状态,有向边代表迁移.

定义 1. 一个标记迁移系统是一个四元组$(S,s_0,A,\rightarrow)$,其中,

- S 表示有限的状态集合;
- $s_0\in S$ 表示初始状态;
- A 表示有限的标记集合(有时候也称为动作);
- $\rightarrow\subseteq S\times A\times S$ 表示标记转移关系.

一般把$(s_1,a,s_2)\in\rightarrow$写成 $s_1\xrightarrow{a}s_2$.一个 LTS 定义一个计算框架.这里,S 表示系统可能开始的状态集合,L 表示系统可以执行的动作集合,A 表示系统迁移的动作集合.

定义 2. 一个双重标记迁移系统(DLTS)是由一个五元组成$(S,s_0,L,A,\rightarrow)$,其中,$(S,s_0,A,\rightarrow)$是一个 LTS,并且 $L:S\rightarrow\{0,1\}$是一个标号函数.

简单标记迁移系统可以表示为:对于所有的状态 s,s'和标号 a,b,如果 $s\xrightarrow{a}s'$ 并且 $s\xrightarrow{b}s'$,则有 $a=b$.一个确定型的 LTS 表示如果有 $s\xrightarrow{a}s_1$ 并且 $s\xrightarrow{a}s_2$,那么 $s_1=s_2$.即,一个状态只有独一无二的迁移动作到达另一个状态.我们定义 $Init(s):\{a\in A\mid \exists t\in S:s\xrightarrow{a}t\}$ 为通过状态 s 出发可执行的一个初始状态集合.

定义 3. 二元关系 $R\subseteq S\times S$ 称为互模拟关系,如果对任意的状态对(s,t),当 $s\ R\ t$ 成立时满足以下条件.

- 若 $s\xrightarrow{a}s'$,则存在某个 t'满足 $t\xrightarrow{a}t'$,并且 $s'\ R\ t'$;
- 若 $t\xrightarrow{a}t'$,则在某个 s'满足 $s\xrightarrow{a}s'$,并且 $s'\ R\ t'$.

一个由所有互模拟状态对组合成的最大的互模拟等价关系叫做互模拟等价关系.

2 互模拟验证算法

给定一个 LTS 和两个状态 s 与 t,一个自然的问题是问 s 与 t 是否互模拟.Paige 和 Tarjan[9]设计了一种分割精化算法,生成给定状态的所有互模拟等价关系类.所以如果 s 和 t 属于相同的等价类,那么他们是互模拟的.这个算法是全局的,它需要知道整个系统的状态,包括那些无关的状态,通过检查所有的状态以及它们的后续状态

是否在同一个等价类中,来不断地把状态集合划分.Fernandez 和 Mounier[6]提出了一种名为 on the fly 的算法,这是一种只需要验证相关 s 和 t 的转移行为的算法.基本思想如下:在通常情况下,s 和 t 分别是两个 LTS 如 L_1 和 L_2 的初始状态.通过深度优先搜索(DFS)遍历两个 LTS 的乘积,表示为 $L_1||L_2$.经过遍历,如果复合状态 $s'||t'$被访问但还未被分析,我们假设 s'和 t'互模拟并继续进行 DFS 遍历.如果两个状态再遍历之后被证明是非互模拟,则原先的假设不成立,继而进行另一次 DFS 遍历.此次的遍历只能证明两个状态不是互模拟.在比较糟糕的情况下,每经历一次 DFS 只能验证一对状态不是互模拟,因此会出现很多重复的情况.为了解决这个问题,Du 和 Deng[7]提出了一种准局部(quasi-local)算法.

准局部算法的主要思想如下:给定两个 LTS 如 L_1 和 L_2[6],首先我们通过 on the fly 的遍历思想,利用两个迁移系统的积构造成一个新的有向图,如果 s 和 t 有不同的初始动作,则我们将 $L_1||L_2$ 中的状态 $s||t$ 标记为 0;反之标记为 1.显然:如果 $s||t$ 被标记为 0,则 s 和 t 一定不是互模拟的;如果被标记为 1,我们暂时还不能确定状态 s 和 t 为互模拟.于是,下一步将标记 0 扩散到每一个不是互模拟的状态对.扩散过程为后退行为.

例如:如果 $s\|t\xrightarrow{a}s'\|t'$ 存在唯一的迁移动作 a 进行从 $s||t$ 到 $s'||t'$的迁移,而且 $s'||t'$已经被标记为 0,那么我们将标记 0 从 $s'||t'$扩散到 $s||t$.其思想就是:如果只有一个 a 动作从 s(或者 t)出发使得 $s\xrightarrow{a}s'$ (或者 $t\xrightarrow{a}t'$)但是 s'不能与 t'进行互模拟,则 s 和 t 也不是互模拟的.一般情况下,$s||t$ 存在一个或多个迁移动作,所以为了适应更加复杂的情况,就需要设计更为合适的数据结构来对标记 0 进行扩散.最后,标记扩散停止的情况有以下两种:(i) 到达初始状态 $s_0||t_0$;(ii) 没有到达初始状态,但是也不存在可以扩散标记 0 的前继状态.第 1 种情况中,$s_0||t_0$ 被标记为 0,表示 s_0 和 t_0 不是互模拟的;第 2 种情况中,$s_0||t_0$ 保持原来的标记 1,表示 s_0 和 t_0 是互模拟的.

实现算法主要用到两个数据结构.

- 一个栈 St 存储所有 0 标记的状态,只要 St 不为空,我们每次移去栈顶的元素,然后开始一轮从它出发的 0 标记扩散过程.每次得到一个新的 0 标记的状态,并把它压入栈 St 中;
- 定义一个三维数组 $Ar_1[k,i,l]$,对于转移 $s_k\xrightarrow{a}s_i$,存在多个可匹配的候选转移(candidate transition) $t_l\xrightarrow{a}t_j$.当我们开始检测互模拟的时候,需要再建立一个三维数组 Ar_2 来存放 $t_l\xrightarrow{a}t_j$ 的候选转移.原算法[7]可用于解决简单的 LTS,因为对于固定的两个状态 s_k 和 s_i,最多只有一个动作为 a 的转移,如 $s_k\xrightarrow{a}s_i$,因此使用三维数组就足够了.

3 准局部算法的扩展

为了使生成的算法[7]不仅仅只适用于简单的 LTS,即:两个给定的状态,可能存在不只一个的迁移关系,我们将三维数组 Ar_1 增加一维,利用四维数组来计算,第四维把迁移动作也考虑进去.例如,$Ar_1[k,i,l,a]$存储对应于从 s_k 到 s_i 的动作为 a 的迁移,存在从 t_l 出发的候选迁移的个数.Ar_2 也做相似的设置.然而,对于大型的 LTS 而言存在上百万的状态,使用多元数组的方式会使得程序较难执行,并且会占用较大的空间.在执行算法时也会容易出现内存溢出.幸运的是,我们观察到,数组 Ar_1 和数组 Ar_2 在通常情况下是稀疏的,所以可以在实现算法的时候,利用哈希表替代四元组.哈希表会使算法执行更快、更能节约内存.

扩展后的准局部算法见算法 1.

算法 1. The optimized quasi-local algorithm.

1: construct the DLTS $L:=S||_RT$ and initialize Ar_1 and Ar_2
2: **if** $L(s_0||t_0)=0$ **then return** FALSE
3: **end if**
4: initialize $St:=\varnothing$ and $L'=L$
5: perform a width first traversal of L and push a pair (i,j) into St whenever
$L(s_i||t_j)=0$
6: **while** $St:=\varnothing$ **do**
7: $(i,j):=pop(St)$

8: **for all** $(s_k||t_l)\in Pred(s_i||t_j)$ with $L'(s_k||t_l)=1$ **do**
9: decrease both $Ar_1[k,i,l,a]$ and $Ar_2[l,j,k,a]$ by 1
10: **if** $Ar_1[k,i,l,a]=0$ or $Ar_2[l,j,k,a]=0$ **then**
11: set $L'(s_k||t_l)=0$
12: **if** $(k,l)=(0,0)$ **then**
13: **return** FALSE
14: **end if**
15: push (k,l) into St
16: **end if**
17: **end for**
18: **end while**
19: **return** TRUE

修改后的算法与原先的大致相同[7],主要区别在于前面介绍的两个四维数组和在实际实现算法过程中使用哈希表.Ar_1的初始值就是迁移 $s_k \xrightarrow{a} s_i$ 被用来导出 $s_k \| t_l \xrightarrow{a} s_i \| t_j$ 形式的次数.Ar_2也用同样的方式初始化.直观来说:如果 LTS 中存在迁移 $s_k \| t_l \xrightarrow{a} s_i \| t_j$,则存在一个迁移 $t_l \xrightarrow{a} t_j$ 能被 t_l 模拟迁移 $s_k \xrightarrow{a} s_i$.在对 0 标记的扩散中,如果 $s_i||t_j$ 被标记为 0,则 $Ar_1[k,i,l,a]$的值减去 1.这里的关键是:如果 s_i 与 t_j 并不互模拟,则 $t_l \xrightarrow{a} t_j$ 并不是一个合适的模仿 $s_k \xrightarrow{a} s_i$ 的迁移.因此,可以用来模仿迁移 $s_k \xrightarrow{a} s_i$ 的候选迁移个数减去 1.同样的,如果 $s_k \xrightarrow{a} s_i$ 不能用于模仿迁移 $t_l \xrightarrow{a} t_j$ 则 $Ar_2[l,j,k,a]$也减去 1.如果 $Ar_1[k,i,l,a]$和 $Ar_2[l,j,k,a]$都变为 0,我们就可以将 $s_k||t_l$ 标记为 0.用这种方法把标记 0 从状态 $s_i||t_j$ 扩散到状态 $s_k||t_l$.在伪代码的第 5 行,我们用宽度优先搜索遍历来找到所有的标记为 0 的状态.事实上,我们也可以利用深度优先搜索遍历来替换.后文表 1 中将给出两个搜索遍历的比较.

用双重标记迁移系统证明互模拟等价的方法 Du 和 Deng 已经给予正确性证明.以下是修改后的算法例子验证,由于修改后的算法,可以验证一般的迁移系统,于是下面给出的两个系统为一般的迁移系统(如图 1 所示).

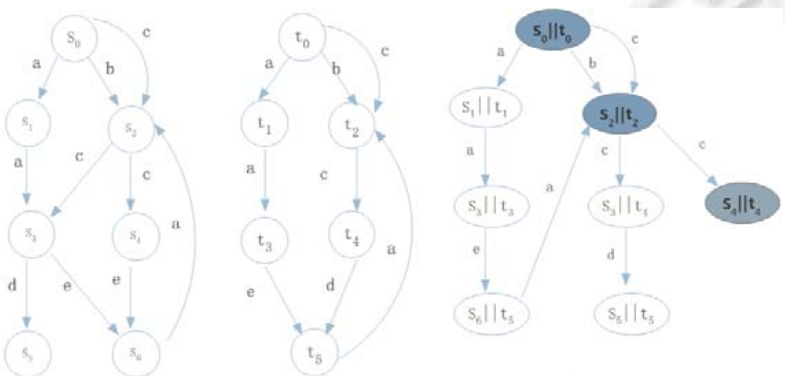

Fig.1 DLTS example
图 1 DLTS 的例子

图 1(a)、图 1(b)为 S 和 T 的两个标记迁移系统,用这两个系统构建一个 DLTS 系统图 1(c).因为 s_4 可以执行动作 e 而 t_4 不可以,所以将状态$(s_4||t_4)$标记为 0,即 t_4 不模拟 s_4.而状态$(s_2||t_2)$是状态$(s_3||t_4)$唯一的一个父节点,因为系统 S 中 $s_2 \xrightarrow{c} s_4$,而在系统 T 中只有 $t_2 \xrightarrow{c} t_4$ 能够匹配,所以 $Ar_1[s_2,s_4,t_2,c]$的值为 1,因为 t_4 不能模拟 s_4,则将 $Ar_1[s_2,s_4,t_2,c]$减 1 得 $Ar_1[s_2,s_4,t_2,c]$为 0,即状态$(s_2||t_2)$标记为 0.以同样的方式对 0 进行扩散,$Ar_1[s_0,s_2,t_0,c]$由于 t_0 能

做 c 动作的只有一个,所以值为 1,$Ar_1[s_0,s_2,t_0,b]$的值也为 1;因为$(s_2||t_2)$标记为 0,所以 t_2 不能模拟 s_2,所以 $Ar_1[s_0,s_2,t_0,c]$与 $Ar_1[s_0,s_2,t_0,b]$减 1 为 0,初始状态$(s_0||t_0)$被标记为 0.则判定系统 T 不能模拟系统 S.

反之,观察系统 S 是否模拟系统 T,以上述同样的方法构建 DLTS,调换状态 s 与 t 的位置即可,以同样的方式标记 0,并做扩散,最后可得状态$(s_0||t_0)$为稳定为 1,则 S 能模拟 T.所以系统 S 与 T 不能互模拟.

扩展后的算法与扩展前的算法除了可以验证标记迁移系统之外,还增加了系统验证的范围.前人已按照原始算法,利用 Java 进行实现,但实现的算法可验证的系统最大迁移量为 10 000,并且只做了 4 组实验,这 4 组实验并无具体数据由来,无法确保实验的正确性.扩展后的算法经过分析改正以及数据结构的扩展处理,可以验证百万的迁移量(具体参见第 4 节);同时,实验数据为 CADP 权威发布的数据,具有一定的可靠性.

4 算法的实现与比较

令 L_1 和 L_2 是两个 LTS,其初始状态分别为 s_0 和 t_0.局部算法和准局部算法都可以验证 s_0 是否与 t_0 是互摸拟的.假设系统 L_1(或 L_2)的状态个数为 n_1(或 n_2),并用 m_1(或 m_2)表示迁移个数.最糟糕的情况下,局部算法的时间复杂度为 $O(n_1^2n_2^2)$.准局部算法的时间复杂度为 $O(m_1m_2)$.对于简单的 LTS, $m_i \leqslant n_i^2$ 其中,i=1,2.准局部算法有很好的时间复杂度,但是对于一般的 LTS 并不一定是如此.

除了理论分析,我们比较两种算法的实际效率.我们已经在 Java 中实现算法.实验环境如下:Windows 7 professioal;Intel(R) Core(TM) i7-4790 CPU 3.60GHz;RAM: 8.00GB.

我们的实验数据以 VLTS 为基准[8],这是一个标记迁移系统的集合.VLTS 的基准获取是来自不同的通信协议和并发系统,都是对应于现实工业系统中的例子.对于表 1 中运用到的每个例子,我们首先从 CADP[10]中下载 BCG 格式的 LTS,然后运用 CADP 自带的分离精化方法,使例子最小化,从而得到一个最简化的 LTS 与原来的 LTS 是互摸拟的.CADP 是为同步并发系统设计的一个很优秀的工具,我们运用实现的两个算法来验证这两个 LTS,表 1 给出经过处理的各例子运行的时间.

Table 1 The comparison of the quasi-local and local algorithms

表 1 准局部算法和局部算法比较

Data	Determinism	n_1	m_1	n_2	m_2	Quasi-Local (DFS) (s)	Quasi-Local (WFS) (s)	Local (DFS) (s)	Local (WFS) (s)
Vasy_386_1171		113	150	113	150	0.003	0.003	0.004	0.005
Vasy_1112_5290	D	265	1 300	265	1 300	0.006	0.008	0.017	0.013
Vasy_0_1		289	1 224	9	20	0.015	0.014	0.020	0.027
Vasy_1_4		1 183	4 464	28	59	0.011	0.011	0.022	0.031
Cwi_1_2		1 952	2 387	1 132	1 432	0.068	0.070	0.087	0.384
Vasy_720_390	D	3 292	116 910	3 292	116 910	1.026	0.961	0.239	2.806
Vasy_574_13561	D	3 577	16 168	3 577	16 168	0.040	0.038	0.085	0.186
Cwi_3_14		3 996	14 552	62	61	0.030	0.027	0.035	0.076
Vasy_5_9		5 486	9 676	145	284	0.035	0.037	0.047	0.087
Vasy_8_24		8 879	24 411	416	1 193	0.089	0.095	0.114	1.185
Vasy_8_38	D	8 921	38 424	219	838	0.064	0.065	0.085	0.219
Vasy_10_56	D	10 849	56 156	2 112	11 372	0.077	0.078	0.216	1.076
Vasy_18_73		18 746	73 043	4 087	16 444	1.390	1.866	6.907	82.446
Vasy_25_25	D	25 217	25 216	25 217	25 216	0.059	0.062	0.098	0.126
Cwi_2165_8723		31 906	144 475	31 906	144 475	0.865	0.918	0.337	3.816
Vasy_40_60	D	40 006	60 007	40 006	60 007	0.110	0.103	3.760	0.201
Vasy_66_1302		66 929	1 302 664	66 929	1 302 664	6.944	6.722	2.420	213.088
Vasy_69_520		69 754	520 633	69 754	520 633	1.248	1.268	0.656	185.364
Vasy_83_325		83 436	325 584	83 436	325 584	6.925	6.250	8.585	3.51608e+03
Vasy_157_297	D	157 604	297 000	4 289	13 642	1.819	1.841	0.719	256.684
Vasy_164_1619		164 865	1 619 204	1 136	3 952	5.843	5.644	8.425	1.40526e+03
Vasy_166_651		166 464	651 168	83 436	325 584	15.942	16.473	25.613	2.60108e+03

注意:这里提供的例子并不是简单的 LTS,这意味着我们不能使用原来的准局部算法[7]

对于每个种算法,我们都使用了深度优先搜索遍历(DFS)和宽度优先搜索遍历(WFS)两种方式来实现.DFS 和 WFS 区别在于验证状态时的顺序,并不会影响算法的正确性.对于局部算法,我们发现 DFS 比 WFS 更加适用,具体两者的比较见表 1.在局部算法中,DFS 直接遍历要判断的两个状态的后续继状态对,我们立即就可以得到

这两个状态是否互模拟的结果;然而 WFS 并不能直接就访问后继的状态对,而是访问兄弟节点上的状态对,所以不能快速地得到两个状态是否互模拟的结果.然而对于准局部算法来说,两种搜索遍历仅仅是用来构建 DLTS 并不会对验证速度产生影响,因此准局部算法的两种遍历结果时间上几乎相同;见表 1 中的第 7 列、第 8 列所示.

比较 DFS 实现的两个算法,从表中的第 7 列、第 9 列可以看出,准局部算法有明显的优势.在我们运行的 22 个例子中,有 17 个例子(其余 5 个例子在表 1 中标记为灰色)显示准局部算法比局部算法时间效率更为优秀.特别是对于确定型的 LTS(在表 1 中第 2 列标记为 D),准局部算法优于局部算法(见表 1 第 6 列、第 8 列).我们还修改了两个算法用于检查模拟关系.具体数据可参见 https://github.com/zhengxiaolin123/bisimulation.

5 弱互模拟

在实际的实例中,我们有时候会提取到两个系统并不全为强互模拟的情况,也有弱互摸拟.为了验证的丰富性,我们对给定的 LTS 进行饱和处理,使算法也能验证弱互模拟.

定义 4. 令$(S,s_0,A,\rightarrow)$为一个标记迁移系统,$\tau\in A$ 为不可见迁移动作.二元关系 $R\subseteq A\times A$ 是一个弱模拟关系,如果对于任意的状态对(s,t),当 $s\ R\ t$ 成立时满足以下条件:

- 若 $s\xrightarrow{a}s'$,则存在 $t'\in S$,使得 $t\xrightarrow{\tau^*}\circ\xrightarrow{a}\circ\xrightarrow{\tau^*}t'$ 并且$(s',t')\in R$,其中,$a\neq\tau$;
- 如果 $s\xrightarrow{\tau}s'$,则 $t\xrightarrow{\tau^*}t'$ 并且$(s',t')\in R$.

如果 R 和它的逆关系 R^{-1} 都是弱模拟关系,那么 R 是弱互模拟关系.

由定义可以观察到:为验证弱互模拟关系,我们可以先对给定的 LTS 做一个饱和(saturation)处理,然后运用前面介绍的互模拟验证算法.所谓饱和处理,就是把原来的迁移系统$(S,s_0,A,\rightarrow)$转化为新的系统$(S,s_0,A,\Rightarrow)$,其中,$\Rightarrow$的定义如下:

- 如果 $a\neq\tau$,那么 $s\overset{a}{\Rightarrow}s'$ 当且仅当 $s\xrightarrow{\tau^*}\circ\xrightarrow{a}\circ\xrightarrow{\tau^*}s'$;
- $s\overset{\tau}{\Rightarrow}s'$ 当且仅当 $s\xrightarrow{\tau^*}s'$.

即:我们利用 Java 程序将不可见迁移动作 τ进行处理,将迁移状态 $t\xrightarrow{\tau^*}\circ\xrightarrow{a}\circ\xrightarrow{\tau^*}t'$ 转换成 $t\xrightarrow{a}t'$,将 $s\xrightarrow{\tau^*}s'$ 转换为 $s\xrightarrow{\tau}s'$,转换后的标记迁移系统可用于已实现的 on the fly 算法和 quasi-local 算法.

我们通过以上定义对获取到的 LTS 进行饱和处理,随即运用于两个算法之中进行互模拟检测.为了使我们实现的算法能运用于现实的例证中,我们用提取实例生成的 LTS 进行实验,表 2 中备注表明了各例子名称,具体可参照 CADP.这些实例都由 CADP 提供的工具进行处理,生成 LTS,再由我们根据弱互摸拟定义对生成的 LTS 进行转换,然后用于实现好的算法中验证.将实例生成 LTS 的方式有很多种,比如:Firace 语言就是将实际的系统装换为 Fiacre 语言,并且利用 Tina 等工具进行转换.CADP 中也提供了 LOTOS 等转换为 LTS 的方法(实验中给出的实例就是经由 LOTOS 转换生成).将转换好的 LTS 进行饱和处理,就可运用于我们所实现的算法之中.表 2 也给出了 CADP 处理互摸拟的时间效率,CADP 采用的编写方式为 C 语言,它用于处理弱互摸拟还是比较有优势的.

我们以比特交换协议(alternating bit protocol,简称 ABP)为例子[17],比特交换协议是在数据链路层上运行的一个简单的网络协议,它使用 FIFO 语义传输丢失或损坏的消息.分析 ABP 说明规范,将其转换为 LOTOS 语言.CADP 在 Demo_Example 的 demo2 里面已经给出了 ABP 说明规范转换为 LOTOS 语言,我们利用 CADP 工具提取 LOTOS 文件并将它利用 CAESAR 转换成 LTS,即 bcg 格式,再将 bcg 格式提取出来转化为 txt 格式.我们再将 txt 进行饱和处理,去除不可见的状态,这是因为我们具体实现的 ABP 是一个不含有不可见状态的 LTS.处理好的 LTS 为 1 161 个状态、456 个迁移量、10 个迁移动作.

Table 2 Verification of weak bisimilarity

表 2 验证弱互模拟关系

	n_1	m_1	n_2	m_2	Java (s)	CADP (s)	备注
Demo_17	13	17	14	24	0.301	0.27	Distributed leader election algorithms specified in LOTOS
Demo_09	981	2 552	23	40	0.312	0.28	INRES protocol
Demo_16	1 373	1 640	47	73	1.732	0.544	Philips' bounded retransmission protocol verified using ACTL temporal logic formulas
Vasy_8_38	8 921	38 424	219	838	0.919	1.137	
Vasy_157_297	157 604	297 000	4 289	13 642	5.785	2.975	
Vasy_8_24	8 879	24 411	416	1 193	10.575	6.306	
Demo_01	1 177	3 904	6	10	12.527	6.351	
Vasy_10_56	10 849	56 156	2 112	11 372	9.368	6.757	
Vasy_166_651	166 464	651 168	83 436	325 584	92.296	56.383	
Vasy_18_73	188 746	73 043	219	838	95.302	90.66	

Fiacre 是一种用于实时系统的高级描述语言,它能将程序转换 LTS,我们利用 Fiacre 实现 ABP,.参考 Fiacre 网站提供的 ABP 实现方法[19].

abp.fcr

```
type seqno is bool
type packet is seqno
process buffer
        [ii: in packet, oo: out packet]
        is
        states idle
        var buff: queue 1 of packet:={||},
        pkt: packet
        from idle
            select
                    ii? pkt;
                    on not (full buff);
                    buff:=enqueue(buff,pkt);
                     to idle
                []
                     on not (empty buff);
                     oo! first buff;
                     buff:=dequeue buff;
                     to idle
                []
                     wait [0,1];
                     on not (empty buff);
                     buff:=dequeue buff;
                    # lost;
                     to idle
              end
process sender
        [mbuff: out packet, abuff: in packet
```

```
29:              is
30:              states idle, send, waita
31:              var ssn,n: seqno:=false
32:            from idle
33:                   to waita
34:           from send
35:                 mbuff! ssn;
36:                 to waita
37:           from waita
38:                 select
39:                 abuff? n;
40:                 if n=ssn then
41:                       ssn:=not ssn
42:                  end;
43:                 to idle
44:            []
45:                   wait [4,5];
46:                   to send
47:            end
48: process receiver
49:            [mbuff: in packet,abuff: out packet]
50:            is
51:            states rcve, ack
52:            var rsn: seqno:=false,
53:          m: packet:=true
54:         from rcve
55:                 mbuff? m;
56:               if m=rsn then
57:                 rsn:=not rsn;
58:               to ack
59:         else
60:              to ack
61:         end
62:       from ack
63:              abuff! m;
64:              to rcve
65: component abp
66:            is
67:            port minp: in out packet in [0,0],
68:            mout: in out packet in [0,1],
69:            ainp: in out packet in [0,2],
70:            aout: in out packet in [0,1]
```

```
71:     par * in
72:         sender[minp,aout]
73:     ||buffer[minp,mout]
74:     ||buffer[ainp,aout]
75:     ||receiver[mout,ainp]
76:     End
77: abp
```

我们将实现好的伪代码在已安装的 Fiacre,Tina 环境中利用 Fiacre 工具,使用语句 frac abp.fcr abp.tts 生成一个 tts 包,这个 tts 包里面包含 abp.c,abp.h,abp.ltl,abp.ndr,abp.det.然后,我们利用 Tina 工具,使用语句:tina—a abp.ndr abp.aut(abp.bcg)将 fcr 文件转换为 aut 文件或者 bcg 文件,再利用 CADP 将其转换为 LTS.我们以同样的方式进行处理 LTS 文件,处理好的文件有 61 个状态、283 个迁移量以及 10 个迁移动作.

我们将说明规范生成的 LOTOS 与利用 Fiacre 语言、根据说明规范实现的 ABP.用我们已经实现的算法进行验证,发现 Fiacre 生成的 LTS 能模拟说明规范 LOTOS 语言生成的 LTS,时间效率为 0.016s.而 ABP 说明规范的 LOTOS 并不能模拟我们所实现的 ABP,所以两个不是互模拟关系.我们实现的互模拟算法也验证了这一点,从而可以证明我们实现的 4 种算法可以用于验证两段程序是否可以模拟或者互模拟来验证实现某种规范算法的正确性.

生成 LTS 还有一种比较常见的方法是还包括 LTSA 工具来生成,利用 FSP 语言编写.lts 文件,可用 ABP 作为例子[16]已经给出.可利用 CADP 生成文件形式的 LTS 或者利用 LTSA 进行图形转换.

所以只要分析系统将系统转换为相应的语言,再将其转换为 LTS,再用实现的算法来验证是否互模拟或者模拟.现实研究中的意义在于可以验证转换器、编译器等的正确性.Julio C.Peralta 等人[18]将 SIGNAL 转换成 Fiacre 语言,并将 Fiacre 利用 CADP 转换成 LTS.同时,利用编译器将 SIGNAL 生成的 C 语言,将 C 语言转换为 Fiacre 语句,以同样的方法转换为 LTS.比较这两个 LTS 为互模拟.从而验证了互模拟的编译器的正确性.

6 总结与展望

我们扩展了 Du 和 Deng 的准局部算法,使其不再只能验证简单的 LTS,还可验证一般的 LTS.用 Java 实现准局部算法并与局部算法进行比较,根据实验数据表明,准局部算法比局部算法更为优秀.我们也修改了算法,让其能验证模拟的系统.最后用于实例中测试,并扩展处理弱互摸拟关系.我们的实验数据以 VLTS 为基准,实例验证的数据由 CADP 提供测试用例.

近来,Groote 和 Wijs[11]提出了一种有效的全局算法,用于验证分支互模拟,重点在于辅助数据结构的使用.Wijs[12]提出了采用并行的 GPU 加速算法.Dalsgaard 等人[13]已经发明了一种分布式算法,计算稳定的依赖图.目前还不清楚这种相似的思想是否可以用于准局部算法.

Kundu[14]倡导了一种验证互摸拟的高级合成技术和定理证明,以此引发了一系列的研究工作[15],我们期待它可以用来扩展我们的准局部算法.

References:

[1] Park D. Concurrency and automata on infinite sequences. In: Proc. of the GI-Conf. on Theoretical Computer Science. 1981. 167−183. [doi: 10.1007/BFb0017309]

[2] Milner R. Communication and Concurrency. Prentice-Hall, Inc., 1989.

[3] Sangiorgi D. Introduction to Bisimulation and Coinduction. Cambridge University Press, 2011.

[4] Dovier A, Piazza C, Policriti A. An efficient algorithm for computing bisimulation equivalence. Theoretical Computer Science, 2002,311(1-3):221−256. [doi: 10.1016/S0304-3975(03)00361-X]

[5] Fisler K, Vardi MY. Bisimulation minimization and symbolic model checking. Formal Methods in System Design, 2002,21(1): 39−78. [doi: 10.1023/A:1016091902809]

[6] Fernandez JC, Mounier L. Verifying bisimulations on the fly. In: Proc. of the 3rd Int'l Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols. North-Holland Publishing Co., 1990. 95−110.

[7] Du WJ, Deng YX. A quasi-local algorithm for checking bisimilarity. In: Proc. of the IEEE Int'l Conf. on Computer Science and Automation Engineering. 2011. 1−5. [doi: 10.1109/CSAE.2011.5952411]

[8] The VLTS benchmark suite. http://cadp.inria.fr/resources/vlts/

[9] Valmari A. Simple bisimilarity minimization in $O(m\log n)$ time. Applications and Theory of Petri Nets, 2010,105(3):319−339. [doi: 10.3233/FI-2010-369]

[10] Garavel H, Lang F, Mateescu R, Serwe W. CADP 2011: A toolbox for the construction and analysis of distributed processes. Software Tools for Technology Transfer, 2013,15(2):89−107. [doi: 10.1007/s10009-012-0244-z]

[11] Groote JF, Wijs A. An $O(m\log n)$ algorithm for stuttering equivalence and branching bisimulation. In: Proc. of the 22nd Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. LNCS 9636, Springer-Verlag, 2016. 607−624. [doi: 10.1007/978-3-662-49674-9_40]

[12] Wijs A. GPU accelerated strong and branching bisimilarity checking. In: Proc. of the 21st Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. LNCS 9035, Springer-Verlag, 2015. 368−383. [doi: 10.1007/978-3-662-46681-0_29]

[13] Dalsgaard AE, Enevoldsen S, Larsen KG, Srba J. Distributed computation of fixed points on dependency graphs. In: Proc. of the 2nd Int'l Symp. on Dependable Software Engineering: Theories, Tools, and Applications. LNCS 9984, Springer-Verlag, 2016. 197−212. [doi: 10.1007/978-3-319-47677-3_13]

[14] Kundu S, Lerner S, Gupta R. Validating high-level synthesis. In: Proc. of the 20th Int'l Conf. on Computer Aided Verification. LNCS 5123, Springer-Verlag, 2008. 459−472. [doi: 10.1007/978-3-540-70545-1_44]

[15] Hao KC, Ray S, Xie F. Equivalence checking for function pipelining in behavioral synthesis. In: Proc. of the DATE 2014. European Design and Automation Association, 2014. 1−6. [doi: 10.7873/DATE.2014.163]

[16] http://www.doc.ic.ac.uk/~jnm/LTSdocumention/AB_example.html

[17] Wikipedia. https://en.wikipedia.org/wiki/Alternating_bit_protocol

[18] Peralta JC, Gautier T, Besnard L, Guernic PL. LTS for Translation Validation of (multi-clocked) SIGNAL Specifications. IEEE, 2010. [doi: 10.1109/MEMCOD.2010.5558632]

[19] Fiacre. http://projects.laas.fr/fiacre/

郑晓琳(1993－),女,福建莆田人,硕士,主要研究领域为形式化方法.

付辰(1991－),男,学士,主要研究领域为模型检测.

邓玉欣(1978－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为形式化方法,程序语义.

雷国庆(1990－),男,硕士,主要研究领域为形式化方法.