ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.20, No.6, June 2009, pp.1444−1456
doi: 10.3724/SP.J.1001.2009.03352

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

参数化系统安全性的启发式符号验证*

杨秋松[1,2], 李明树[1,3+]

[1](中国科学院 软件研究所 互联网软件技术实验室,北京 100190)

[2](中国科学院 研究生院,北京 100049)

[3](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)

Heuristic Symbolic Verification of Safety Properties for Parameterized Systems

YANG Qiu-Song[1,2], LI Ming-Shu[1,3+]

[1](Laboratory for Internet Software Technologies, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](Graduate University, The Chinese Academy of Sciences, Beijing 100049, China)

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

+ Corresponding author: E-mail: mingshu@iscas.ac.cn, http://www.iscas.ac.cn

Yang QS, Li MS. Heuristic symbolic verification of safety properties for parameterized systems. *Journal of Software*, 2009,20(6):1444−1456. http://www.jos.org.cn/1000-9825/3352.htm

Abstract: A parameterized system is a system that involves numerous instantiations of the same finite-state process, and depends on a parameter which defines its size. The backward reachability analysis has been widely used for verifying parameterized systems against safety properties modeled as a set of upward-closed sets. As in the finite-state case, the verification of parameterized systems also faces the state explosion problem and the success of model checking depends on the data structure used for representing a set of states. Several constraint-based approaches have been proposed to symbolically represent upward-closed sets with infinite states. But those approaches are still facing the symbolic state explosion problem or the containment problem, i.e. to decide whether a set of concrete states represented by one set of constraints is a subset of another set of constraints, which is co-NP complete. As a result, those examples investigated in the literature would be considered of negligible size in finite-state model checking. This paper presents several heuristic rules specific to parameterized systems that can help to mitigate the problem. Experimental results show that the efficiency is significantly improved and the heuristic algorithm is several orders of magnitude faster than the original one in certain cases.

Key words: parameterized system; safety property; upward-closed set; heuristic search; symbolic verification

摘 要: 参数化系统(paramterized system)是指包含特定有限状态进程多个实例的并发系统,其中的参数是指系统内进程实例的数目,即系统的规模.反向可达性分析(backward reachability analysis)已被广泛用于验证参数化系统是否满足以向上封闭(upward-closed)集合表示的安全性(safety property).与有限状态系统验证相类似,参数化系统的验证同样也面临着状态爆炸(state explosion)问题,并且模型检测算法的有效性依赖于如何采用有效的数据结构表示

* Supported by the National Natural Science Foundation of China under Grant No.60573082 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.2006AA01Z185 (国家高技术研究发展计划(863))

Received 2007-11-12; Accepted 2008-03-28

状态集合.针对表示无穷状态的向上封闭集合,研究人员提出了多种基于约束(constraint-based)的符号表示方法.但这些方法依然面临着符号状态爆炸(symbolic state explosion)问题或者其包含判定问题,即判断一个约束条件集合符号化表示的实际状态集合是否为另一约束条件集合所对应的状态集合的子集,是Co-NP完全问题.因此,虽然有限状态验证技术能够验证一些具有一定规模的问题,但现有针对参数化系统的验证方法所能解决的问题的规模较为有限,需要近一步提高模型检测算法的效率.针对参数化系统提出了加快反向可达性分析的多个启发式规则,实验结果表明,这些启发式规则可以使算法的效率提高几个数量级,从而有助于解决现有参数化系统验证方法所存在的问题.

关键词: 参数化系统;安全性;向上封闭集合;启发式搜索;符号验证

中图法分类号: TP301 **文献标识码:** A

很多应用领域中广泛存在着一类并发系统:在系统运行过程中,需要动态地创建进程实例以处理外部请求或者进行后台计算.虽然每个进程的状态空间是有限的,或者其抽象(abstraction)的状态空间是有限的,但很难在运行之前准确地预测系统内进程实例数目的上界;即使可以预测出该上界,而所得到的预测值往往很大,导致很难直接应用现有的有限状态验证技术进行分析.为了描述这类系统,参数化系统这一概念被提了出来,它是指包含特定有限状态进程多个实例的并发系统,其中的参数是指系统内进程实例的数目,也即系统的规模[1].

对于给定的参数化系统和用某种逻辑表示的性质,若称该性质在给定的系统中成立(或者说该系统满足给定的性质),则是指该性质在包含任意多个进程实例的参数化系统中都成立.因此,这类问题的实质是对具有无穷状态空间的系统进行验证[2].对于一般的参数化系统,这类问题是不可判定的,而且这一结论不依赖于描述性质时所使用的逻辑[3].但对于特殊的一类参数化系统,其安全性验证是可判定的,在这类系统中,所有动态创建的进程实例需要完全相同,即系统中只有一个进程可被用来动态地创建新实例[4−8].其中,一类主要判定过程是基于反向可达性分析算法[4,8],其优点在于可以判定某些在前向可达分析[5]算法下不能终止的问题.Javier[6]证明了对于确定性(deterministic)广播协议(broadcast protocol),虽然其活性(liveness property)是不可判定的,但由于系统状态集合上所具有的良拟序关系(well-quasi-ordering),使得其安全性的验证是可判定的,并且基于有限状态自动机的安全性可以表示为向上封闭集合,进而通过参数化系统反向可达性分析进行验证.

与有限状态系统的验证相类似,参数化系统的验证同样也面临着状态爆炸问题,并且模型检测算法的有效性依赖于如何采用有效的数据结构表示状态集合.由于向上封闭集合表示一个具有无穷元素的集合,同时在反向可达性分析过程中所得到的集合也都是无穷集合,因此需要用符号方法来表示这些集合.线性约束方法被用来符号化表示参数化系统分析和验证过程中产生的向上封闭集合[9−11],其基本思想是,用一系列线性约束条件来表示一个或者多个向上封闭集合,而每个线性约束条件对应了向上封闭集合的一个最小元(minimal element).这些方法所存在的主要问题是,虽然引入了符号方法表示状态集合,但依然面临着状态爆炸问题(不妨称其为符号状态爆炸),NA-约束(non-addition constraints) [10]的数目和Sharing Tree[11]所包含路径的数目与约束条件集合中常量最大值之间呈指数关系.而另外一些数据结构上的包含判定问题,即判断一个约束条件集合符号化表示的实际状态集合是否为另一约束条件集合所对应的状态集合的子集,是Co-NP完全问题.比如,对于包含相加操作的约束条件和DV-constraints的包含判定问题[10]以及Sharing tree的包含判定问题[11]都是Co-NP完全问题.而包含判定问题是反向可达性分析中最为基本的操作之一,该条件成立时,表明反向可达性分析到达了一个不动点(fixed-point),即此时算法可以终止.因此,虽然有限状态验证技术能够验证一些具有一定规模的问题,但现有针对参数化系统的验证方法所能解决的问题的规模比较有限,需要进一步提高模型检测算法的效率.

在本文中,我们针对参数化系统提出了一些启发式规则,一系列实验表明,通过相应的启发式规则可以极大地改善算法的效率.与现有方法相比,对某些问题的验证可以提高几个数量级,从而可以缓解现有方法因符号状态爆炸或者包含判定问题所具有的 Co-NP 复杂性导致的效率问题.本文第 1 节给出了参数化系统的基本定义,包括语法、语义和安全性的验证.第 2 节针对非确定性参数化系统,我们给出了一个反向可达性分析算法.第 3 节介绍针对参数化系统的启发式规则.第 4 节通过一系列实验表明所提出的启发式规则可以显著地改善算法的效率.第 5 节和第 6 节分别给出了相关工作和结论.

1 参数化系统

1.1 语 法

不失一般性,本文所考虑的参数系统包含两部分:控制器(monitor)和可以动态创建新实例的有限状态进程(在不引起歧义的情况下,简称为进程;同时,在不引起混淆的情况下,进程实例也称作进程).控制器和进程之间通过同步或者广播方式进行通信,而两个进程实例之间不进行直接通信.参数化系统为一个三元组$\langle \Sigma\cup\{\tau\},\mathcal{M},\mathcal{P}\rangle$.其中,$\Sigma$表示系统的基本动作(action)集合,$\tau$表示内部行为,$\mathcal{M}$表示控制器对应的有限状态自动机,它定义为四元组$\langle Q_{\mathcal{M}},\Sigma\times\{?,!,??,!!\}\cup\{\tau\},s^0_{\mathcal{M}},\delta_{\mathcal{M}}\rangle$,其中,$Q_{\mathcal{M}}$表示状态集合;$\Sigma\times\{?,!,??,!!\}\cup\{\tau\}$表示自动机的字母表,包含同步输入和输出动作$\Sigma\times\{?,!\}$、广播输入与输出动作$\Sigma\times\{??,!!\}$和内部行为$\tau$;$s^0_{\mathcal{M}}$表示$\mathcal{M}$的初始状态;$\delta_{\mathcal{M}}\subset Q_{\mathcal{M}}\times\{\Sigma\times\{?,!,??,!!\}\cup\{\tau\}\}\times Q_{\mathcal{M}}$表示状态转移函数.$\mathcal{P}$表示可以动态创建新实例的有限状态进程,定义为四元组$\langle Q_{\mathcal{P}},\Sigma\times\{?,!,??,!!\}\cup\{\tau\},s^0_{\mathcal{P}},\delta_{\mathcal{P}}\rangle$,其中每个元素的定义与$\mathcal{M}$相类似.

作为一个例子,图 1 给出了一个非确定性的负载均衡系统[10].开始时,控制器处于idle状态,而存在任意数目的进程处于req状态.当控制器位于idle状态时,一个进程可以通过执行*request*!或者*release*!动作从req状态转换到use状态,或者从use状态转换到req状态.对于所有位于use状态的进程,当控制器执行动作*swap_out*!!时将会不确定地转换到low或者high状态.对于所有具有较高优先级的处于high状态的进程,当控制器执行动作*swap_in*!!时将会返回到use状态;而具有较低优先级的处于low状态的进程将返回到req状态,需要继续进行申请才能再次处于use状态.

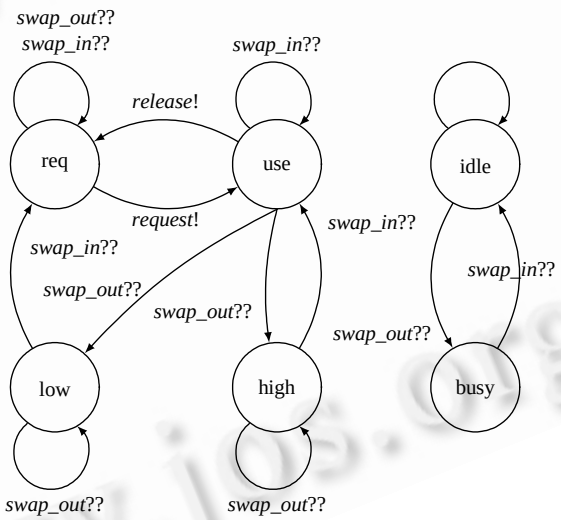

Fig.1 Load balancing example

图 1 负载均衡实例

1.2 语 义

参数化系统的配置(全局状态)表示为$\boldsymbol{c}$,为一向量$\langle s_{\mathcal{M}},n_1,...,n_m\rangle$.其中,$s_{\mathcal{M}}$表示控制器的当前状态,$m$表示进程$\mathcal{P}$的状态数,而$n_i$表示处于状态$s_i\in Q_{\mathcal{P}}$的进程数.同时,$\boldsymbol{c}_{\mathcal{M}}$表示$\boldsymbol{c}$中的$s_{\mathcal{M}}$元素,$\boldsymbol{c}(i)$表示元素$n_i$,而$\boldsymbol{c}_{\mathcal{P}}$表示配置中的向量$\langle n_1,\ldots,n_m\rangle$.$\boldsymbol{c}(i:n_i')$表示在新向量中用$n_i'$代替$\boldsymbol{c}$中的$n_i$,而其他部分与$\boldsymbol{c}$相同.参数化系统的配置集合表示为$C$,是集合$Q_{\mathcal{M}}\times\{\mathbb{N}\cup\{\omega\}\}^m$的一个子集,其中,$\omega$表示比任何指定自然数大的整数.因此,参数化系统的语义δ描述了一个系统可以执行的转换,它为$C\times\{\Sigma\cup\{\tau\}\}\times C$的子集,其中每个元素具有$(\boldsymbol{c},a,\boldsymbol{c}')$的形式,表示系统通过执行动作$a(a?$和$a!$或者$a??$和$a!!)$从配置$\boldsymbol{c}$转换到配置$\boldsymbol{c}'$,也可以表示为$\boldsymbol{c}\rightarrow^a\boldsymbol{c}'$.另外,$\boldsymbol{c}\rightarrow\boldsymbol{c}'$表示存在$a\in\Sigma$,使得$\boldsymbol{c}\rightarrow^a\boldsymbol{c}'\in\delta$.$\delta$可以由如下结构化规则定义:如果$s_{\mathcal{M}}\rightarrow^\tau s'_{\mathcal{M}}$,则对于所有满足$\boldsymbol{c}_{\mathcal{M}}=s_{\mathcal{M}}$,$\boldsymbol{c}'_{\mathcal{M}}=s'_{\mathcal{M}}$和$\boldsymbol{c}_{\mathcal{P}}=\boldsymbol{c}'_{\mathcal{P}}$的$\boldsymbol{c}$和$\boldsymbol{c}'$,

均有$\boldsymbol{c}\rightarrow^{\tau}\boldsymbol{c}'$成立.如果$s_i\rightarrow^{\tau}s_j$,则对于所有满足$\boldsymbol{c}(i)>0$ 和$\boldsymbol{c}'=\boldsymbol{c}(i:\boldsymbol{c}(i)-1,j:\boldsymbol{c}(j)+1)$的$\boldsymbol{c}$和$\boldsymbol{c}'$,均有$\boldsymbol{c}\rightarrow^{\tau}\boldsymbol{c}'$成立.如果$s_{\mathcal{M}}\rightarrow^{a?}s'_{\mathcal{M}}$且$s_i\rightarrow^{a!}s_j$,则对于所有满足$\boldsymbol{c}(i)>0$ 和除$\boldsymbol{c}_{\mathcal{M}}=s_{\mathcal{M}},\boldsymbol{c}'_{\mathcal{M}}=s'_{\mathcal{M}}$外$\boldsymbol{c}'=\boldsymbol{c}(i:\boldsymbol{c}(i)-1,j:\boldsymbol{c}(j)+1)$的$\boldsymbol{c}$和$\boldsymbol{c}'$,均有$\boldsymbol{c}\rightarrow^{a}\boldsymbol{c}'$成立.如果$s_{\mathcal{M}}\rightarrow^{a!}s'_{\mathcal{M}}$且$s_i\rightarrow^{a?}s_j$,则对于所有满足$\boldsymbol{c}(i)>0$和除$\boldsymbol{c}_{\mathcal{M}}=s_{\mathcal{M}},\boldsymbol{c}'_{\mathcal{M}}=s'_{\mathcal{M}}$外$\boldsymbol{c}'=\boldsymbol{c}(i:\boldsymbol{c}(i)-1,j:\boldsymbol{c}(j)+1)$的$\boldsymbol{c}$和$\boldsymbol{c}'$,均有$\boldsymbol{c}\rightarrow^{a}\boldsymbol{c}'$成立.如果$s_{\mathcal{M}}\rightarrow^{a!!}s'_{\mathcal{M}}$,则$\boldsymbol{c}\rightarrow^{a}\boldsymbol{c}'$在$\boldsymbol{c}$和$\boldsymbol{c}'$满足如下条件时成立,该条件考虑了参数化系统的不确定性:

$$V_i=\left\{\boldsymbol{v}\,|\,\boldsymbol{v}=\sum_{\{s_k|s_i\xrightarrow{a??}s_k\}}t_k\boldsymbol{u}_k,\boldsymbol{c}(i)=\sum_{\{s_k|s_i\xrightarrow{a??}s_k\}}t_k,t_k\geq 0\right\},1\leq i\leq m,$$

$$\boldsymbol{c}'\in\left\{\langle s'_{\mathcal{M}},\boldsymbol{v}\rangle\,|\,\boldsymbol{v}=\boldsymbol{v}_1+\ldots+\boldsymbol{v}_m,\boldsymbol{v}_i\in V_i\right\},\text{其中,}|\,\boldsymbol{u}_k\,|=m\text{且}\boldsymbol{u}_k(j)=\begin{cases}0, & \text{if } i\neq j\\ 1, & \text{if } i=j\end{cases}.$$

1.3 安全性验证

对于参数化系统的验证,我们主要考虑可以规约为从向上封闭集合出发进行反向可达性分析的问题.对于集合X上的拟序关系$\preceq$(自反和传递)和集合$S\subseteq X$,若对于任意$y\in X$满足$\exists x\in S$使得$x\preceq y$时总有$y\in S$成立,则集合S被称为向上封闭集合.对于任意$x\in X$,$\uparrow x$表示集合$\{y\,|\,y\succeq x\}$,相应地,$\uparrow S=\bigcup_{s\in S}\uparrow s$.对于一个向上封闭集合$S$,它的基(basis)为$S$的一个子集$S_b$且满足$S=\uparrow S_b$.对于给定的参数化系统,其上的拟序关系$(\preceq,C)$定义为:$\langle\boldsymbol{c},\boldsymbol{c}'\rangle\in\preceq$当且仅当$\boldsymbol{c}_{\mathcal{M}}=\boldsymbol{c}'_{\mathcal{M}}$且对于任意$1\leq i\leq m$满足$\boldsymbol{c}(i)\leq\boldsymbol{c}'(i)$.

若集合B表示系统运行过程中不期望出现的配置集合,并且通过反向可达性分析不能找到一个初始配置$\boldsymbol{c}_0$满足$\boldsymbol{c}_0\in BackReach(B)$(其中,$BackReach(B)$为从集合$B$出发,通过反向可达性分析可以到达的配置集合),则系统满足安全性B.对于图 1 的负载均衡系统,其互斥安全性定义为:当控制器处于busy状态时,不存在任何一个进程处于 use 状态.该安全性的否定可以用向上封闭集合$\uparrow B=\uparrow\{\langle s_{busy},n_{req}=0,n_{use}=1,n_{low}=0,n_{high}=0\rangle\}$来表示,当不存在某个初始配置$\boldsymbol{c}_0\in BackReach(\uparrow B)$时,则表明负载均衡系统满足该互斥安全性.

2 反向可达性分析

2.1 基本方法

定义 1(良拟序关系). 对于集合 X 上的拟序关系$\preceq$(自反和传递),当对于任何由 X 中元素构成的无穷序列$x_0,x_1,x_2,\ldots$,存在索引$i<j$使得$x_i\leq x_j$时,则该关系被称作良拟序关系.

定义 2(良构转换系统). 对于转换系统$\mathcal{S}=\langle S,\rightarrow,\preceq\rangle$,若在$S$上定义的拟序关系$\preceq\subseteq S\times S$满足:拟序关系$\preceq$是一个良拟序,$\preceq$与$\rightarrow$满足向上兼容(upward compatible),即对于所有$s_1\preceq t_1$和转换$s_1\rightarrow s_2$,存在着一个系列转换$t_1\xrightarrow{*}t_2$使得$s_2\preceq t_2$.

定理 1(参数化系统的良构性). 对于第 1 节定义的非确定性参数化系统$B=\langle\Sigma\cup\{\tau\},\mathcal{M},\mathcal{P}\rangle$,若其上的拟序关系$(\preceq,C)$定义为$\langle\boldsymbol{c},\boldsymbol{c}'\rangle\in\preceq$当且仅当$\boldsymbol{c}_{\mathcal{M}}=\boldsymbol{c}'_{\mathcal{M}}$且对于任意$1\leq i\leq m$满足$\boldsymbol{c}(i)\leq\boldsymbol{c}'(i)$,则该系统为良构转换系统.

证明:首先$(\preceq,C)$是良拟序,该结论来自于Dickson定理[12],该定理指出,若$v_1,v_2,\ldots$为具有$\mathbb{N}^k$形式元素的无穷序列,则存在$i<j$使得$v_i\preceq v_j$.在兼容性方面,对于任意$\boldsymbol{c}_1$和$\boldsymbol{c}_2$,若满足$\boldsymbol{c}_1\preceq\boldsymbol{c}'$且$\boldsymbol{c}_1\rightarrow^{a}\boldsymbol{c}_2$,则存在$\boldsymbol{c}''\in\boldsymbol{C}$使得$\boldsymbol{c}'\rightarrow^{a}\boldsymbol{c}''$且$\boldsymbol{c}_2\preceq\boldsymbol{c}''$,这是因为具有更多进程实例的系统可以模拟具有较少进程实例的系统. □

因此,可以利用针对无穷状态系统和良构转换系统所提出的反向可达性分析算法[4,8]对参数化系统进行验证.该算法的终止依赖于配置间的良拟序关系,否则将会导致一个无穷序列且其中不存在任何两个配置$\boldsymbol{c}_i,\boldsymbol{c}_j(i<j)$使得$\boldsymbol{c}_i\preceq\boldsymbol{c}_j$.根据参数化系统的特点,图 2 给出了一种广度优先的反向可达性分析算法,该算法到达一个初始配置时就停止,即在反向可达集中存在一个配置$\boldsymbol{c}$使得$\boldsymbol{c}\preceq\left\langle s^0_{\mathcal{M}},n_{s^0_{\mathcal{P}}}=\omega,0,\ldots,0\right\rangle$.其中的 PRE 算法,对于给定的参数系统和向上封闭集合,返回该集合的直接前驱(immediate predessor).虽然图 2 是以广度优先的形式给出的,也可以构造相应的深度优先算法.

```
BackReach (badStates)     // a set of bad configurations
1: visitedStates:=∅
2: workSet:=badStates
3: while workSet is not empty do
4:   predecessors:=PRE(workSet)
5:   visitedStates:=visitedStates∪workSet
6:   workSet:=predecessors
7:   for each s∈workSet do
8:     if s ⪯ ⟨s^0_M, n_{s_P} = ω,0,...,0⟩ then
9:       print counterexample_shortest
10:    end if
11:    if ∃s_1 ∈ visistedStates such that s_1 ⪯ s then
12:      delete s from workSet
13:    end if
14:  end for
15: end while
end
```

Fig.2 Backward reachability analysis

图 2 反向可达性分析

2.2 PRE算法

根据第 1 节关于安全性的描述,反向可达性分析过程中的初始集合(系统运行过程中不期望出现的配置集合)为一个向上封闭集合.对于参数化系统而言,该集合及其前驱一般是无穷集合,因此需要引入符号方法以有效地表示这些集合.若一个拟序关系是良序的,则基于该关系定义的向上封闭集合的基为一个有限集合;否则存在一个无穷严格递减序列($\boldsymbol{c}_1 \succ \boldsymbol{c}_2 \succ \boldsymbol{c}_3 \succ ...$),从而与该拟序关系为良拟序相矛盾.同时,对于向上封闭集合S,若$\boldsymbol{c}_1 \in PRE(S)$(即$\exists \boldsymbol{c}_2 \in S$使得$\boldsymbol{c}_1 \to \boldsymbol{c}_2$),则对于任意满足 $\boldsymbol{c}_1 \preceq \boldsymbol{c}_3$ 的$\boldsymbol{c}_3$,由于参数化系统的良构性,将会存在$\boldsymbol{c}_4 \in S$使得$\boldsymbol{c}_3 \to \boldsymbol{c}_4$成立且满足 $\boldsymbol{c}_2 \preceq \boldsymbol{c}_4$,即$\boldsymbol{c}_3 \in PRE(S)$.因此,若一个集合为向上封闭集合,则其在参数化系统中的直接前驱也为向上封闭集合,同时具有一个有限的基.

针对第 1 节给出的非确定性、进程和控制器之间通过同步或者广播方式进行通信以及两个进程之间禁止直接通信的参数化系统,算法PRE返回一个向上封闭集合的直接前驱,图 3 给出了算法PRE的详细伪码.它分两种情况构造一个向上封闭集合的前驱,即控制器和特定进程之间通过同步方式进行交互和控制器通过广播输出动作影响所有的进程.每一个向上封闭集合通过其有限基来进行描述,而有限基中的每个元素对应着一个NA-约束.一个NA-约束是一系列具有$n_i \geq k$形式的原子约束(atomic constraint)的合取,其中,$n_i \in \{n_1,...,n_m\}$,而k是一个非负整数.

若 $\boldsymbol{c}$ 为当前迭代中反向可达集中的元素,不失一般性,假设控制器执行动作 $a!$,而存在某个进程执行动作 $a?$,即该进程和控制器之间通过执行动作 a 实现同步.如图 4 所示,$\boldsymbol{c}$ 的前驱可以分两种情况讨论:$\boldsymbol{c}(l)>0$ 且存在转换 $s_k \to^{a?} s_l \in \delta_{\mathcal{P}}$,则$\boldsymbol{c}(k{:}n_k{+}1, l{:}n_l{-}1)$即为$\boldsymbol{c}$的直接前驱;另外一个转换 $s_i \to^{a?} s_j \in \delta_{\mathcal{P}}$ 也可以响应控制器的动作$a!$,但由于$\boldsymbol{c}(j)=0$,使得$\boldsymbol{c}(i{:}n_i{+}1, j{:}n_j{-}1)$因$n_j-1=-1$ 而不可能是$\boldsymbol{c}$的直接前驱.另一方面,由于反向可达性分析的当前集合为向上封闭集合且 $\boldsymbol{c} \preceq \boldsymbol{c}(j:n_j+1)$,因此,$\boldsymbol{c}(j{:}n_j{+}1)$属于当前反向可达集且$\boldsymbol{c}(i{:}n_i{+}1) \to^{a?} \boldsymbol{c}(j{:}n_j{+}1)$,进而$\boldsymbol{c}(i{:}n_i{+}1)$也为$\boldsymbol{c}$的直接前驱.

如图 5 所示,当控制器执行广播输出动作$a!!$时,所有处于状态s_i和s_k的进程将会转换到状态s_j;而对于某些处于不能执行动作$a??$的状态(比如状态s_1和s_m)的进程,则它们的状态不发生改变.在算法PRE中,先为每一个状态 $s \in Q_{\mathcal{P}}$ 构造一个向量集合.若一个状态(比如s_1)不存在标记为$a??$的扇入边(incoming edges),则其对应的集合为$\{\mathbf{0}(r{:}n_r)\}$,其中,$\mathbf{0}$ 为m维零向量,r为该状态的索引;若存在标记为$a??$的扇入边,则需要在该状态所对应的集合内列出在不考虑其他状态情况下的所有可能前驱.对于图 5 中的状态n_j,其所对应的集合为$\{\mathbf{0}(i{:}a, k{:}b, j{:}c) | a+b+c=n_j\}$.若 V_{s_i} 为每个状态 $s_i \in Q_{\mathcal{P}}$ 所对应的集合,则$\boldsymbol{c}$的直接前驱为

$$\{\boldsymbol{c}' = \langle \boldsymbol{c}'_{\mathcal{MT}}, \boldsymbol{c}'_{\mathcal{ST}} \rangle \mid \boldsymbol{c}'_{\mathcal{MT}} \to^{a!!} \boldsymbol{c}_{\mathcal{MT}}, \boldsymbol{c}'_{\mathcal{ST}} = \sum_{i=1}^{m} \boldsymbol{v}_{s_i}, \boldsymbol{v}_{s_i} \in V_{s_i}\}.$$

PRE (*currentBasis*) // an upward-closed set's finite basis
1: *preSet*:=∅
2: **for each** $\boldsymbol{c}\in$*currentBasis*
3: **for each** $s'_{\mathcal{M}} \rightarrow^{a!(\text{or } a?)} s_{\mathcal{M}}$ and $s_i \rightarrow^{a?(\text{or } a!)} s_j$ do
4: **let** $\boldsymbol{c}' := \boldsymbol{c}, \boldsymbol{c}'_{\mathcal{M}} = s'_{\mathcal{M}}$
5: **if** $\boldsymbol{c}(j)>0$ **then**
6: $\boldsymbol{c}'(j):=\boldsymbol{c}'(j)-1$
7: **end if**
8: $\boldsymbol{c}'(i):=\boldsymbol{c}'(i)+1$
9: **if** $\exists \boldsymbol{c}'' \in$ *preSet* such that $\boldsymbol{c}'' \preceq \boldsymbol{c}'$ **then**
10: **continue**
11: **end if**
12: **add** $\boldsymbol{c}'$ to *preSet*
13: **end for**
14: **for each** $s'_{\mathcal{M}} \rightarrow^{a!!} s_{\mathcal{M}}$ **do**
15: **if** $\exists i\,(1 \le i \le m)$ such that $\boldsymbol{c}(i) > 0$, $\exists s_k \in Q_{\mathcal{P}}$ such that $s_k \ne s_i$ and $s_i \rightarrow^{a??} s_k \in \delta_{\mathcal{P}}$, and $\nexists s_l \in Q_{\mathcal{P}}$ such that $s_l \rightarrow^{a??} s_i \in \delta_{\mathcal{P}}$ **then**
16: **continue** // c is not resulted from broadcasting
17: **end if**
18: **for each** s_i and $\{s_{i_1} \rightarrow^{a??} s_i, ..., s_{i_k} \rightarrow^{a??} s_i\}$ **do**
19: **if** $\{i_1,\ldots,i_k\}$ is not empty **then**
20: let $V_{s_i} := \{\boldsymbol{v} \mid \boldsymbol{v}(l) = 0, \text{if } l \notin \{i_1,...,i_k\}$ and
21: $\boldsymbol{v}(i_1)+\ldots+\boldsymbol{v}(i_k)=\boldsymbol{c}(i)\}$
22: **else**
23: $V_{s_i} := \{0,...,n_{s_i} = \boldsymbol{c}(i),...,0\}$
24: **end if**
25: **end for**
26: **for each** $combination\{\boldsymbol{v}_1,...,\boldsymbol{v}_m \mid \boldsymbol{v}_i \in C_{s_i}\}$ **do**
27: let $\mathcal{M} := \sum_{i=1}^{m} \boldsymbol{v}_i, \boldsymbol{c}' := \langle s'_{\mathcal{M}}, \mathcal{M} \rangle$
28: **if** there exists $\boldsymbol{c}'' \in$ *preSet* such that $\boldsymbol{c}'' \preceq \boldsymbol{c}'$ **then**
29: **continue**
30: **end if**
31: **add** $\boldsymbol{c}'$ to *preSet*
32: **end for**
33: **end for**
34: **end for**
35: **return** *preSet*
End

Fig.3 PRE: Return predecessors of a given upward-closed set

图 3 PRE 算法:返回指定向上封闭集合的直接前驱

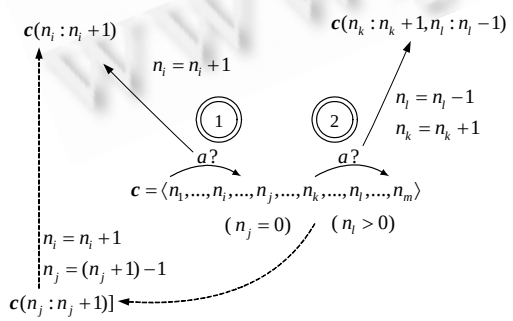

Fig.4 Predecessors of rendezvous actions

图 4 同步动作的前驱

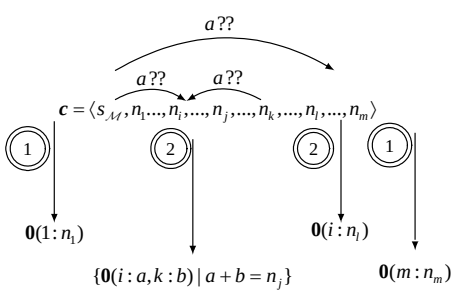

Fig.5 Predecessors of broadcast actions

图 5 广播动作的前驱

定理 2(PRE算法的正确性). 对于用有限基S_1表示的向上封闭集合$\uparrow S_1$,若$PRE(S_1)$返回的其前驱的基为S_2,则对于所有$\boldsymbol{e}\in\uparrow S_2$存在$\boldsymbol{e}'\in\uparrow S_1$使得$\boldsymbol{e}\rightarrow\boldsymbol{e}'$,且对于所有$\boldsymbol{e}\in\uparrow S_1$和使得$\boldsymbol{e}'\rightarrow\boldsymbol{e}$成立的配置$\boldsymbol{e}'$,都有$\boldsymbol{e}'\in\uparrow S_2$成立.

证明:$(\forall \boldsymbol{e}(\boldsymbol{e}\in\uparrow S_2)\Rightarrow\exists \boldsymbol{e}'(\boldsymbol{e}'\in\uparrow S_1\wedge\boldsymbol{e}\rightarrow\boldsymbol{e}'))$由参数化系统的良构性可以得出该结论成立.

$(\forall \boldsymbol{e}(\boldsymbol{e}\in\uparrow S_1\wedge\forall \boldsymbol{e}'(\boldsymbol{e}'\rightarrow\boldsymbol{e}\Rightarrow\boldsymbol{e}'\in\uparrow S_2)))$对于所有$\boldsymbol{e}\in\uparrow S_1$,若存在配置$\boldsymbol{e}'$使得$\boldsymbol{e}'\rightarrow\boldsymbol{e}$,则存在动作$a$使得$\boldsymbol{e}'\rightarrow^a\boldsymbol{e}$,根据$a$分别为同步和广播动作时分两种情况讨论:若$a$为同步动作,则根据向上封闭集合基的定义,存在$\boldsymbol{e}_1\in S_1$使得$\boldsymbol{e}_1\preceq\boldsymbol{e}$,根据图 4 所示的规则,算法PRE将会在$S_2$中添加配置$\boldsymbol{e}_2$满足$\boldsymbol{e}_2\rightarrow^a\boldsymbol{e}_1$,同时根据图 4 所示的计数器变化规律(若$s_i\rightarrow^a s_j$,则$\boldsymbol{c}'(i):=\boldsymbol{c}'(i)+1$,而$\boldsymbol{c}'(j)=\boldsymbol{c}(j)(\boldsymbol{c}(j)=0)$或者$\boldsymbol{c}'(j)=\boldsymbol{c}(j)-1(\boldsymbol{c}(j)>0)$)和$\boldsymbol{e}_1\preceq\boldsymbol{e}$,可以得到$\boldsymbol{e}_2\preceq\boldsymbol{e}'$,即$\boldsymbol{e}'\in\uparrow S_2$;若$a$为一个广播动作,根据向上封闭集合基的定义,则存在$\boldsymbol{e}_1\in S_1$使得$\boldsymbol{e}_1\preceq\boldsymbol{e}$,算法PRE将会根据图 5 所示的规则向$S_2$中添加一组$\boldsymbol{e}_1$对应的前驱.由于$\boldsymbol{e}_1\preceq\boldsymbol{e}$和图 5 所示的规则,则对于$\boldsymbol{e}$的任意前驱$\boldsymbol{e}'$(通过执行广播活动$a$),则在$\boldsymbol{e}_1$的前驱中存在$\boldsymbol{e}_2$使得$\boldsymbol{e}_2\preceq\boldsymbol{e}'$,即$\boldsymbol{e}'\in\uparrow S_2$. □

3 启发式规则

由于参数化系统所具有的良构性,上述反向可达性分析算法将会在有限步骤内终止.此时,若某个初始配置属于当前可达集,则相应的安全性在系统中不成立;或者,当遍历了所有反向可以到达的状态空间后,仍不能到达某个初始配置,则该性质成立.对于模型检测算法而言,其空间和时间效率是最为关注的问题之一.对于上面给出的基于 PRE 算法的反向可达性分析算法,一个向上封闭集合被表示为一个或者多个 NA-约束的析取.虽然NA-约束的包含判定问题具有多项式时间复杂度,但是集合$\boldsymbol{pre}_{NA}(\Phi)$的基数(cardinality)为$O(|\Phi|\times a\times C_{n+c}^{c})$(其中,$\boldsymbol{pre}_{NA}(\Phi)$表示NA-约束$\Phi$的直接前驱,$n$和$a$分别为参数化系统的状态数和动作数,而$c$为$\Phi$中的最大常量),将会导致约束条件数目的增长与$c$或者$n$呈指数关系[10].图 6 给出了基于广度优先和深度优先的反向可达性分析算法在负载均衡系统上的运行结果.不同于一般的软件验证,参数化系统验证中某个性质成立,是指该性质在包含任意多个进程实例的参数化系统中都成立,即不依赖于系统的规模(一般是指系统中并发进程的数目).因此,图 6 的横坐标是指具有不同复杂度的安全性,随着序号的增加,性质变得越来越复杂.至于所选的性质和性质的含义,将在后续部分中详细加以讨论.从图 6 可以看出,算法的执行时间和所需的空间随着性质复杂度的提高而显著增加,其中,RC(reachable configurations)是指在反向可达性分析过程中所访问的配置数.

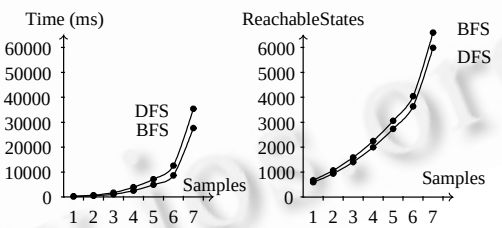

Fig.6 Verification results of the load balancing system

图 6 负载均衡系统的验证结果

为了进一步提高模型检测的效率,本节针对参数化系统提出了可以加快反向可达性分析的启发式规则.基于这些规则,反向可达性分析算法首先展开那些最有希望的配置,计算其直接前驱并加入到当前可达集合.而后,从新的集合中继续选择最有希望的配置并求其直接前驱,直到发现某个初始配置表明性质不成立或者到达一个不动点,则表明性质成立.启发式规则对于给定的配置决定其优先级,进而决定访问配置的顺序,尝试访问那些最有希望的路径以提高算法效率.本文中,启发式规则的评估函数 f 主要根据待选节点与目标节点(初始配置集合)之间的差别计算一个配置的优先级.

3.1 规则H_0

在该规则中,一个配置的优先级取决于其处于状态$s_{\mathcal{P}}^0$(进程$\mathcal{P}$的初始状态)的进程数.对于一个配置$\boldsymbol{c}$而言,当

其满足 $\boldsymbol{c} \preceq \langle s^0_{\mathcal{M}}, n_{s^0_{\mathcal{P}}} = \omega, 0, ..., 0\rangle$ 时,则被称为初始配置.对于给定的配置 $\boldsymbol{c}' = \langle s_{\mathcal{M}}, n_1, ..., n_m\rangle$,图 3 的反向可达性分析算法试图找到一系列的逆向转换 $\rightarrow^{a_1}, ..., \rightarrow^{a_n}$ 和一个初始配置 $\boldsymbol{c}'' = \langle s^0_{\mathcal{M}}, n'_1, 0, ..., 0\rangle$,使得 $\boldsymbol{c}'' \rightarrow^{a_1} ... \rightarrow^{a_n} \boldsymbol{c}'$.根据算法PRE,将会有 $n'_1 = \sum_{i=1}^{m} n_i + \theta$ 成立,其中,θ是在反向可达性分析过程中执行图 4 中标记为 1 的动作的次数.启发式规则H_0的出发点是首先选择具有较大n_1的配置,因此,其评估函数定义为$f(\boldsymbol{c})$=$\boldsymbol{c}(1)$,即处于状态 $s^0_{\mathcal{P}}$ 的进程数.

3.2 规则$\boldsymbol{H_{IH}}$

与同步动作不同,一个广播动作可能导致反向可达性分析过程中当前可达集合的急剧膨胀,特别是当进程$\mathcal{P}$中某个状态可以通过相应的广播输入动作逆向到达多个状态时,将会使得当前可达集合更为迅速地膨胀.以配置 $\boldsymbol{c} = \langle s_{\mathcal{M}}, ..., n_k, ..., n_i = \lambda, ..., n_l, ...\rangle$ 为例,假设对于所有$j(0 \le j \le m, j \ne i)$都有$n_j$=0,并且存在广播动作$a$使得 $s_k \rightarrow^{a??} s_i \in \delta_{\mathcal{P}}$, $s_l \rightarrow^{a??} s_i \in \delta_{\mathcal{P}}$ 和 $s'_{\mathcal{M}} \rightarrow^{a!!} s_{\mathcal{M}}$ 成立.根据算法PRE,下列配置将会作为$\boldsymbol{c}$的直接前驱被加入到当前可达集的直接前驱集合:

$$\begin{gathered}
\langle s'_{\mathcal{M}}, ..., n_k = 0, ..., n_i = \lambda, ..., n_l = 0, ...\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 1, ..., n_i = \lambda - 1, ..., n_l = 0\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 0, ..., n_i = \lambda - 1, ..., n_l = 1\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 2, ..., n_i = \lambda - 2, ..., n_l = 0\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 1, ..., n_i = \lambda - 2, ..., n_l = 1\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 0, ..., n_i = \lambda - 2, ..., n_l = 2\rangle,\\
\cdots,\\
\langle s'_{\mathcal{M}}, ..., n_k = \lambda, ..., n_i = 0, ..., n_l = 0\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = \lambda - 1, ..., n_i = 0, ..., n_l = 1\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 1, ..., n_i = 0, ..., n_l = \lambda - 1\rangle,\\
\langle s'_{\mathcal{M}}, ..., n_k = 0, ..., n_i = 0, ..., n_l = \lambda\rangle.
\end{gathered}$$

上面给出了所有可以通过执行动作 a 而转换到 $\boldsymbol{c}$ 的配置,且不存在任何两个配置 $\boldsymbol{c}'$和 $\boldsymbol{c}''$满足 $\boldsymbol{c}'' \preceq \boldsymbol{c}'$.因此,需要把上面列出的所有配置加入到 $\boldsymbol{c}$ 的直接前驱集合中,集合中配置的数目为 $\sum_{i=0}^{\lambda}(i+1) = (\lambda+1)(\lambda+2)/2$.当$\mathcal{P}$中有更多的状态可以通过动作$a$??转换到状态$s_j$时,该数目将会变得更大.

规则H_{IH}倾向于如下事实:每个前驱对于$\boldsymbol{c}$的影响并不是完全相同的,即处于状态s_i的进程或者是更多地通过状态s_k转换而来,或者是更多地通过状态s_l转换而来,而不是均衡地各有一半分别从s_k和s_l转换而来.因此,该规则倾向于选择具有较大n_k或者n_l的配置作为优先展开的对象.更一般地,对于给定的配置$\boldsymbol{c}$,其优先级定义为 $f(\boldsymbol{c}) = \max_{i=1}^{m}\boldsymbol{c}(i) - \min_{i=1}^{m}\boldsymbol{c}(i)$.

4 实验结果

4.1 实验方法

为了证明启发式规则的有效性,如下 4 个系统将作为本文的实验系统:图 1 所示的负载均衡系统(load balancing,简称LB)、广播协议(broadcast protocol,简称BP)[6]、文件共享访问系统(shared file access,简称SFA)[13]以及CPU互斥访问系统(exclusive CPU access,简称ECU)[12].与文献[13]中的文件共享系统相比,我们只考虑了具有一个文件的简化版本,而原来的系统中有两个文件.

在 LFAF 和 EAC 的进程$\mathcal{P}$中出现了广播输出动作,比如 *lock*!!,*unlock*!!,*reada*!!和 *writea*!!,通过这些动作,进程间可以进行直接通信.而第 1 节关于参数化系统的定义不允许进程间进行直接通信,前面曾提到,该限制不会影响参数化系统的描述能力,通过下述转换可以使得LFAF和EAC符合第 1 节给出的参数化系统的定义.对于进程中的广播输出动作,$s_1 \rightarrow^{a!!} s_2$,可以通过如下规则进行变换:如果控制器为空(即在参数化系统中只有可以动

态创建实例的进程),则在 $Q_{\mathcal{M}}$ 中添加新状态s并且对 $\delta_{\mathcal{P}}$ 中每个同步动作$\rightarrow^{b!}(\rightarrow^{b?})$添加$s\rightarrow^{b?}s(s\rightarrow^{b!}s)$到 $\delta_{\mathcal{M}}$;分别添加新状态t和新动作$siga$到 $Q_{\mathcal{P}}$ 和Σ,以及向 $\delta_{\mathcal{P}}$ 加入新转换$s_1\rightarrow^{siga!}t$和$t\rightarrow^{a??}s_2$;在 $\delta_{\mathcal{P}}$ 中删除转换$s_1\rightarrow^{a!!}s_2$;在 $Q_{\mathcal{M}}$ 中添加新的状态t',并对于除状态t'外 $Q_{\mathcal{M}}$ 内的每个状态s添加转换$s\rightarrow^{siga?}t$和$t\rightarrow^{a!!}s$到 $\delta_{\mathcal{M}}$.

进行必要的转换后,参数化系统的基本信息见表 1.其中,$|Q_{\mathcal{M}}|$和$|Q_{\mathcal{P}}|$分别表示 $Q_{\mathcal{M}}$ 和 $Q_{\mathcal{P}}$ 内的状态数,$|\delta_{\mathcal{M}}|$和$|\delta_{\mathcal{P}}|$分别表示 $\delta_{\mathcal{M}}$ 和 $\delta_{\mathcal{P}}$ 内的转换数.表 2 列出了所要验证的属性以及可以动态调整的参数.除了 ECA 中的$\langle use,wait\rangle$属性成立以外,其他几个属性在各自的系统中是不成立的.

在实验过程中,我们利用 Java 实现了上面针对非确定性参数化系统所提出的广度优先和深度优先反向可达性分析算法,以及引入启发式规则后的反向可达性分析算法,并采集了验证表 2 中每个性质时所有算法运行所需的时间以及所访问的配置数.实验的平台为配置 DualCore T2400@1.83GHz CPU 和 2GB 内存且运行 Windows XP 的 IBM ThinkPad.同时,为了更准确地测量每种算法的运行时间,使用 JNI 技术调用测量 Java 中每个线程执行时间的本地库,而不是测量系统所在 CPU 的运行时间,从而可以尽量避免系统内其他进程的干扰.

Table 1 Parameters of experiment systems

表 1 实验系统的基本参数

System	$\lvert Q_{\mathcal{M}}\rvert$	$\lvert\delta_{\mathcal{M}}\rvert$	$\lvert Q_{\mathcal{P}}\rvert$	$\lvert\delta_{\mathcal{P}}\rvert$
LB	2	4	4	10
BP	3	5	3	9
SFA	3	6	6	16
ECA	6	10	5	6

Table 2 Properties to be verified

表 2 验证的性质

Property	*P*1	*P*2	*P*3	*P*4	*P*5
$\langle low,high\rangle$	⟨5,5⟩	⟨6,6⟩	⟨7,7⟩	⟨8,8⟩	⟨9,9⟩
$\langle c_1,c_2\rangle$	⟨10,10⟩	⟨20,20⟩	⟨30,30⟩	⟨40,40⟩	⟨50,50⟩
$\langle I,S_a\rangle$	⟨2,2⟩	⟨3,3⟩	⟨4,4⟩	⟨5,5⟩	⟨6,6⟩
$\langle use,wait\rangle$	⟨5,5⟩	⟨6,6⟩	⟨7,7⟩	⟨8,8⟩	⟨9,9⟩

4.2 实验结果

为了更清楚地呈现实验结果,我们使用算法间的性能数据比值而不是原始数据来说明算法的效率.其中,性能数据包括每种算法的运行时间和所访问的配置数.

结论 1. 广度优先反向可达性分析算法不一定优于深度优先反向可达性分析算法,反之亦然.

对于图 3 不考虑启发式规则的广度优先反向可达性分析算法(BBFS)及相应的深度优先算法(BDFS),分别对表 1 中的每个系统验证表 2 中相应的性质,并记录每次运行所需的时间和访问的配置数,两种算法的性能比值如图 7 和图 8 所示.

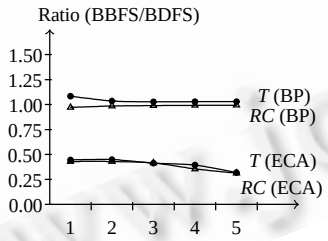

Fig.7 BBFS vs. BDFS (1)

图 7 BBFS 和 BDFS 的对比(1)

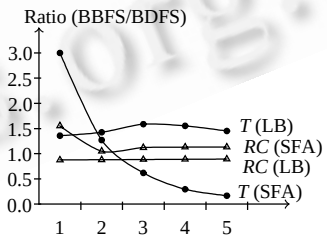

Fig.8 BBFS vs. BDFS (2)

图 8 BBFS 和 BDFS 的对比(2)

在运行时间方面:BBFS 在 ECA 的验证中比 BDFS 算法的效率要高;但 BDFS 算法在 LB 中优于 BBFS 算法;而在 BP 中,它们的效率相差不大.对于图 8 所示的 SFA,BDFS 算法在性质比较简单时具有较高的效率;而 BBFS 与 BDFS 的比值随着性质复杂度的提高,呈现出迅速下降的趋势,从而 BBFS 算法在性质比较复杂时具有较高的效率.在空间方面:BBFS 在 ECA 中所访问的配置数大约只为 BDFS 的一半;而在另外几个系统中,两种算法的空间效率基本相同.

结论 2. 通过引入启发式规则H_{IH}和H_0,可以显著地提高算法的效率.

与蛮力(brute-force)反向搜索算法相比,最佳启发式搜索总是展开被认为最有希望的节点,而不是随机地挑

选或者选择待访问节点列表中的第 1 个或者最后一个.其基本的工作方式与DFS算法相类似,不同的只是节点访问顺序.因此,我们对标准BDFS算法和加入启发式规则的BDFS算法进行了对比,以准确地衡量启发式规则的引入对算法效率的影响.启发式规则H_{IH}的实验结果如图 9 和图 10 所示,对于LB和BP的验证,算法的运行时间比原BBFS算法分别提高了 25%和 150%;而对于SFA,启发式规则的引入使得算法的效率提高了大约 11 倍;在ECA中,虽然效率的改善不是很显著,但与原算法相比,对某些性质的验证可以提高 10%.

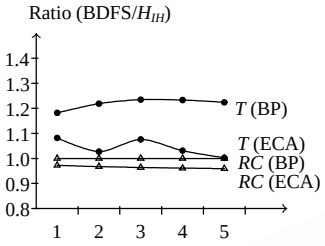

Fig.9 BDFS vs. H_{IH} (1)

图 9 BDFS和H_{IH}的对比(1)

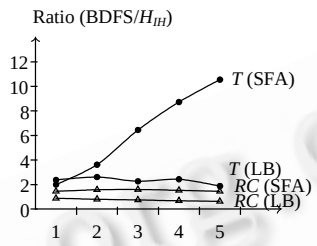

Fig.10 BDFS vs. H_{IH} (2)

图 10 BDFS和H_{IH}的对比(2)

从规则本身来讲,H_0似乎比H_{IH}更为简单和直接,但从图 11 和图 12 的结果来看,前者可以取得更好的效果.在图 12 中,为了能够把取值范围相差很大的曲线画在一起,我们使用了双纵坐标形式,其中,算法运行时间的比值(黑色圆点标记)参考左侧纵坐标,而算法访问配置数的比值(空白的三角标记)参考右侧纵坐标.与原BDFS算法相比,对于LB验证的时间效率可以提高约 90 倍,而所访问的配置数只是原来算法的一半;对于SFA的验证,其时间效率大约提高了 30 倍;与规则H_{IH}相比,BP和ECA的验证效率得到了进一步明显的改善.

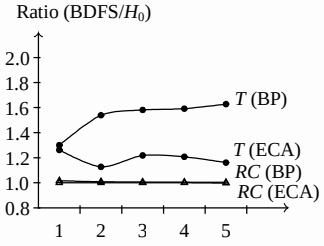

Fig.11 BDFS vs. H_0 (1)

图 11 BDFS和H_0的对比(1)

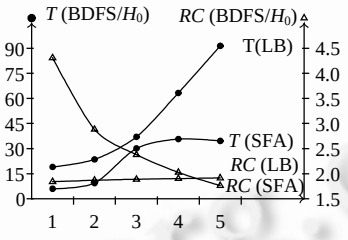

Fig.12 BDFS vs. H_0 (2)

图 12 BDFS和H_0的对比(2)

结论 3. 若采用DV-约束而不是NA-约束表示向上封闭结合,H_{IH}和H_0也可以显著地改善算法的效率.

上述讨论的实验结果是在利用 NA-约束表达向上封闭集合的情况下获得的,而另外一类表示向上封闭集合的典型约束形式是 DV-约束.一个 DV-约束是具有如下形式且包含加号的约束条件:

$$x_{1,1}+\ldots+x_{1,n_1}\geq k_1\wedge\ldots\wedge x_{m,1}+\ldots+x_{m,n_m}\geq k_m,$$

其中,任何一个变量$x_{i,j}$最多出现在一个原子条件中.由于加号操作的引入,使得一个DV-约束可以表达有时需要用一组NA-约束才能表达的条件.虽然DV-约束的包含判定问题为Co-NP完全问题,但其局部包含问题(判断由一个DV-约束表示的实际配置集合是否为另一个DV-约束所对应的集合的子集)具有多项式时间复杂度,而且$\boldsymbol{pre}_{DV}(\Phi)$的基数为$O(|\Phi|\times a\times c^2)$,与$c$之间为多项式关系[10].

与图 3 中的反向可达性分析算法及其启发式算法相对应,我们基于DV-约束实现了支持确定性参数化系统的反向可达性分析算法.如果在参数化系统中引入非确定性,将会使基于 DV-约束的验证过程变得比较复杂,进而失去其所具有的优势.这主要是因为非确定性的引入将会使仿射矩阵和参数系统动作之间的对应关系丢失,从而需要在反向可达性分析过程中不断地进行约束条件的拆分和合并,以使得每一个变量只能出现在一个原子约束中.该转换过程与Giorgio[11]从$\mathcal{U}$公式向析取范式的转换类似,一般来讲,新问题的规模与原有问

题的规模呈指数关系.

从理论上讲,基于DA-约束的方法可以节省更多的内存空间,同时具有更高的时间效率.如图 13 和图 14 所示,基于DA-约束的BDFS算法与基于NA-约束的BDFS算法相比具有更高的空间效率,特别是在SFA中,前者所用的空间有时只是后者的1%;而在时间方面,基于DA-约束的算法在SFA中具有较高的效率,而在BP中则不及基于NA-约束的算法.图 15 给出了启发式规则的引入对于基于DA-约束的算法的改进结果.从图中可以看出,规则H_0和H_{IH}都可以非常显著地改善算法的时间和空间效率,其中,时间效率提高了约 3 000 倍,而相应的空间效率可以提高 40 倍.我们也在另外两个确定性系统BP和SFA上测试了启发式规则对于基于DA-约束的算法的影响,但发现其对这两个系统验证效率的改进效果不是很显著.这主要是因为在SFA中有很多广播动作,在反向可达性分析过程中需要很多回溯,启发式规则可以帮助算法找到最有希望的路径,从而可以显著地提高算法效率,表明本文的启发式规则适合于验证比较复杂的参数化系统.

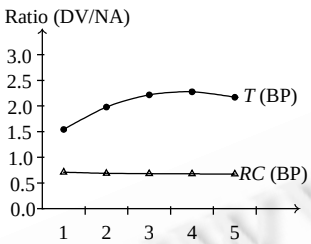

Fig.13 DV vs. NA in BP

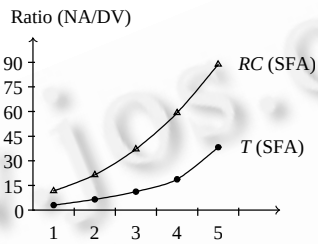

Fig.14 DV vs. NA in SFA

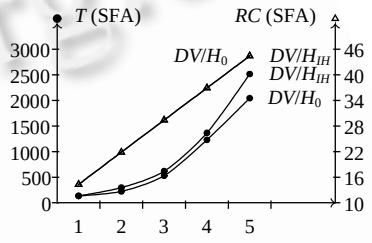

Fig.15 DV vs. DV+*H* in SFA

图 13 DV 和 NA 在 BP 中的对比 图 14 DV 和 NA 在 SFA 中的对比 图 15 DV+H 和 DV 在 SFA 中的对比

5 相关工作

针对良构转换系统和无穷状态系统,Finkel等人[8]和Abdulla等人[4]分别提出了基于反向可达性分析的一般方法.对于给定的向上封闭集合,Javier[6]给出了计算其直接前驱的方法,该方法基于如下两条规则: $\boldsymbol{c}'_{\mathcal{M}} \rightarrow^{a} \boldsymbol{c}_{\mathcal{M}}$ 和 $M_a \cdot \boldsymbol{c}'_{\mathcal{M}} + \boldsymbol{v}_a = \boldsymbol{c}_{\mathcal{M}}$,其中,$\boldsymbol{c}$和$\boldsymbol{c}'$分别属于该向上封闭集合及其直接前驱的基,M_a和$\boldsymbol{v}_a$为与动作a对应的转换矩阵和向量.该方法的主要问题在于第 2 条规则并不是总成立,而且每个输出动作,比如 a!和 a!!,只能在整个系统中最多出现 1 次.而本文在第 2 节给出的反向可达性分析算法不受相应的限制,可以很好地处理参数化系统中的非确定性.

为了表示参数化系统验证过程中所产生的无穷集合,研究者们基于线性约束提出了符号方法以改进模型检测的效率[9−11],但这些方法面临着符号状态爆炸问题或者具有Co-NP时间复杂度的包含判定问题.Bingham提出了基于传统BDD的良构转换系统验证方法[14],该方法的基本思想是,对于具有特定规模(进程数目一定)的良构转换系统,利用基于BDD的模型检测技术对其进行验证,如果性质得到满足,则把系统的规模加大(对于参数化系统而言,进程数目加 1)后重新进行验证.该算法的终止依赖于一个针对特定良构转换系统所提出来的收敛定理,虽然在某些情况下可以改进验证的效率,但与其他方法相比,比如Sharing Tree,其效率有时相差几个数量级.同时,由于收敛定理的需要,只能描述一类很特殊的系统(Nicely Sliceable WSTS,简称δ-NSW).与本文通过启发式规则提高参数化系统验证效率的想法相类似,Delzanno[15]针对可以表示为Petri网的参数化系统提出了一个基于Petri网结构特性(structural invariant)的启发式验证方法.该方法中利用Sharing Tree作为表示向上封闭集合的符号方法,而Sharing Tree的每条路径表示一个NA-约束条件,虽然一些模型算法中的操作(比如求前驱、包含判定)可以直接在Sharing Tree上进行,但Sharing Tree的包含判定问题为Co-NP完全问题,而本文所采用的NA-约束的包含判定问题具有多项式的时间复杂度.下一步的工作将会对基于Petri网结构特性的启发式规则和本文中提出的启发式规则的性能进行比较.

6 结 论

在本文中,针对非确定性参数化系统提出了一种计算给定向上封闭集合直接前驱的算法 PRE.为了简化该算法,在参数化系统的定义中限制参数化系统中进程间的直接通信,而该限制并不影响参数化系统的描述能力.通过在第 4 节给出的规则,可以把进程中的广播输出动作用控制器中的广播输出动作和一组控制器和进程间的同步输入和输出动作来替代.同时,该转换并没有使问题的规模和复杂度显著增加.基于算法PRE,参数化系统的安全性验证可以转化为反向可达性分析,若某个初始配置属于可达集,则相应的安全性在系统中不成立;或者当遍历了所有反向可以到达的状态空间时仍不能到达某个初始配置,则该性质成立.为了进一步提高算法的效率,我们引入了针对参数化系统的启发式规则.一系列的实验结果表明,通过该规则,可以使得算法的效率在某些情况下提高几个数量级.

在三维集成软件开发模型TRISO Model(tridimensional integrated software development model)[16,17]中,软件过程的要素主要包括活动(activity)、角色(actor)和工作产品(artifact),被抽象为表现出特定行为且与外部环境和其他要素进行交互的实体.由于在实际软件过程中通常包含着多个特定实体的实例(比如一个活动被拆分为一组类似的活动、一个角色被分配到多个人或者一类的工作产品可以用同一个实体来刻画),而且由于软件过程执行时每个实体的实例数随着所开发软件规模的改变而不断地发生变化,很难给出系统内每个实体实例数的上界,因此,软件过程也是一类特殊的参数化系统.在文献[18,19]中,我们基于进程代数方法描述了这类软件过程,并对具有特定规模的软件过程系统进行了分析和验证.下一步,我们将把本文所提出的验证方法和启发式规则与已有工作方法结合起来,以更好地描述和分析软件过程,并改进分析算法的效率.

References:

[1] Zuck L, Pnueli A. Model checking and abstraction to the aid of parameterized systems (a survey). Computer Languages, Systems and Structures, 2004,30(3-4):139−169.

[2] Javier E. Verification of systems with an infinite state space. In: Proc. of the Modeling and Verification of Parallel Processes. New York: Springer-Verlag, 2001. 183−186.

[3] Apt KR, Kozen DC. Limits for automatic verification of finite-state concurrent systems. Information Processing Letters, 1986,22(6): 307−309.

[4] Abdulla PA, Cerans K, Jonsson B, Yih-Kuen T. General decidability theorems for infinite-state systems. In: Proc. of the 11th Annual IEEE Symp. on Logic in Computer Science. IEEE Computer Society, 1996. 313−321.

[5] Emerson EA, Namjoshi KS. On model checking for non-deterministic infinite-state systems. In: Proc. of the 13th Annual IEEE Symp. on Logic in Computer Science. IEEE Computer Society, 1998. 70−80.

[6] Javier E, Alain F, Richard M. On the verification of broadcast protocols. In: Proc. of the 14th Annual IEEE Symp. on Logic in Computer Science. IEEE Computer Society, 1999. 352−359.

[7] German SM, Sistla AP. Reasoning about systems with many processes. Journal of the ACM, 1992,39(3):675−735.

[8] Finkel A, Ph S. Well-Structured transition systems everywhere! Theoretical Computer Science, 2001,256(1-2):63−92.

[9] Delzanno G, Podelski A. Constraint-Based deductive model checking. Int'l Journal on Software Tools for Technology Transfer, 2001,3(3):250−270.

[10] Delzanno G. Constraint-Based model checking for parameterized synchronous systems. In: Proc. of the 4th Int'l Workshop on Frontiers of Combining Systems. Springer-Verlag, 2002. 72−86.

[11] Delzanno G, Raskin JF. Symbolic representation of upward-closed sets. In: Proc. of the 6th Int'l Conf. on Tools and Algorithms for Construction and Analysis of Systems: Held as Part of the European Joint Conf. on the Theory and Practice of Software, ETAPS 2000. Springer-Verlag, 2000. 426−440.

[12] Delzanno G. Constraint-Based verification of parameterized cache coherence protocols. Formal Methods in System Design, 2003, 23(3):257−301.

[13] Delzanno G, Esparza J, Podelski A. Constraint-Based analysis of broadcast protocols. In: Proc. of the 13th Int'l Workshop and 8th Annual Conf. of the EACSL on Computer Science Logic. Springer-Verlag, 1999. 72−86.

[14] Bingham JD. A new approach to upward-closed set backward reachability analysis. Electronic Notes in Theoretical Computer Science, 2005,138(3):37−48.

[15] Delzanno G, Raskin JF, Begin LV. Attacking symbolic state explosion. In: Proc. of the 13th Int'l Conf. on Computer Aided Verification. Springer-Verlag, 2001. 298−310.

[16] Li MS. Assessing 3-D integrated software development processes: A new benchmark. In: Wang Q, Pfahl D, Raffo DM, Wernick P, eds. Proc. of the SPW/ProSim 2006. LNCS 3966, Shanghai: Springer-Verlag, 2006. 15−38.

[17] Li MS. Expanding the horizons of software development processes: A 3-D integrated methodology. In: Proc. of the SPW 2005. LNCS 3840, Beijing: Springer-Verlag, 2005. 54−67.

[18] Li MS, Yang Q, Zhai J, Yang G. On mobility of software processes. In: Wang Q, Pfahl D, Raffo DM, Wernick P, eds. Proc. of the SPW/ProSim 2006. LNCS 3966, Shanghai: Springer-Verlag, 2006. 105−114.

[19] Yang Q, Li MS, Wang Q, Yang G, Zhai J, Li J, Hou L, Yang Y. An algebraic approach for managing inconsistencies in software processes. In: Wang Q, Pfahl D, Raffo DM, eds. Proc. of the ICSP 2007. LNCS 4470, Minneapolis: Springer-Verlag, 2007. 121−133.

杨秋松(1977－),男,河北泊头人,博士生,主要研究领域为软件过程方法与技术,并发系统验证.

李明树(1966－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为软件过程方法与技术.