软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2023,34(8):3674−3685 [doi: 10.13328/j.cnki.jos.006615]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

基于非交互式 Petri 网的异步程序验证模型和方法*

吴志文, 李国强

(上海交通大学 软件学院, 上海 200240)

通信作者: 李国强, E-mail: li.g@sjtu.edu.cn

摘　要: 异步程序使用异步非阻塞调用方式来实现程序的并发, 被广泛应用于并行与分布式系统中. 验证异步程序复杂性很高, 无论是安全性还是活性均达到 EXPSPACE 难. 提出一个异步程序的程序模型系统, 并在其上定义两个异步程序上的问题: ϵ 等价性问题和 ϵ 可达性问题. 通过将 3-CNF-SAT 规约到这两个问题, 再将其规约至非交互式 Petri 网的可达性证明两个问题是 NP 完备的. 案例表明, 这两个问题可以解决异步程序上一系列的程序验证问题.

关键词: 异步程序; 非交互式 Petri 网; ϵ 可达性; ϵ 等价性; 可达性

中图法分类号: TP311

中文引用格式: 吴志文, 李国强. 基于非交互式Petri网的异步程序验证模型和方法. 软件学报, 2023, 34(8): 3674–3685. http://www.jos.org.cn/1000-9825/6615.htm

英文引用格式: Wu ZW, Li GQ. Model and Method for Verifying Asynchronous Program Based on Communication-free Petri Net. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3674–3685 (in Chinese). http://www.jos.org.cn/1000-9825/6615.htm

Model and Method for Verifying Asynchronous Program Based on Communication-free Petri Net

WU Zhi-Wen, LI Guo-Qiang

(School of Software, Shanghai Jiao Tong University, Shanghai 200240, China)

Abstract: Asynchronous programs utilize asynchronous non-blocking calls to achieve program concurrency, and they are widely applied in parallel and distributed systems. However, it is very complex to verify asynchronous programs, and the difficulty can be ranked as EXPSPACE in terms of both safety and liveness. This study proposes a program model of asynchronous programs and defines two problems of asynchronous programs, namely, ϵ-equivalence and ϵ-reachability. In addition, the two problems can be proved to be NP-complete by reducing the 3-CNF-SAT to the problems and making them further reduced to the reachability of the communication-free Petri net. The case shows that the two problems can solve the verification problems of asynchronous programs.

Key words: asynchronous program; communication-free Petri net; ϵ-reachability; ϵ-equivalence; reachability

1 引　言

随着计算机多核系统与分布式系统的大规模发展, 异步程序得到了前所未有的广泛应用. 异步程序是一种普遍存在的系统编程风格, 用于管理与环境的并发交互. 异步程序使用异步非阻塞调用方式来实现程序的并发. 这种调用不需要等待函数或者过程返回结果, 可以直接继续执行程序; 而同步阻塞调用需要等待函数或者过程返回结果, 然后才能继续执行程序. 然而编写正确的异步程序也非易事, 对异步程序的程序验证更是一类难题. 由于一个具有多栈的程序模型是图灵完备的[1], 因此通常来说, 异步程序上的验证问题是不可判定的. 除此以外, 不同线程

* 基金项目: 国家自然科学基金 (61872232, 61732013)
本文由“形式化方法与应用”专题特约编辑陈立前副教授、孙猛教授推荐.
收稿时间: 2021-09-03; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-03-24
CNKI 网络首发时间: 2023-02-24

© 中国科学院软件研究所 http://www.jos.org.cn

之间以异步方式通讯, 需要借助不同媒介, 如队列 (queue)、共享缓冲区 (shared buffer, 通常表示为多重集合)、邮箱 (mailbox) 等, 也为验证带来了一定的困难[2]. 因此学术界为了得到异步程序可判定的性质, 通常采用近似或抽象等方法, 对程序模型作出一定限制. 即便如此, 异步程序的程序验证依然困难[3,4].

在异步程序模型的研究中, Sen 等人将异步通讯程序限定为拥有一个无限的程序栈, 要求通讯时程序栈需为空, 提出了多重集下推系统[5], 并将异步程序的安全性问题归约到 Petri 网[6]的可覆盖性问题上. Ganty 等人则在此基础之上, 借助 Parikh 像将异步程序的活性问题归约到 Petri 网上的可达性问题上[7]. Atig 等人在多重下推系统上, 通过禁止在同一个线程上执行的任务之间传输锁, 证明了带有嵌套锁的多线程异步程序上的可达性问题的复杂度是 EXPSPACE-complete[8]. Emmi 等人提出面向事件驱动 (如中断) 的异步程序模型[9], 并通过数据 Petri 网这个 Petri 网的扩展模型的可覆盖性问题来解决事件驱动异步程序的安全性问题. 众所周知, Petri 网是一种状态转移系统, 可以用来描述异步、并发的计算机系统模型. Petri 网的可覆盖性问题的复杂度是 EXPSPACE 完备, Petri 网的可达性问题的复杂度则是 EXPSPACE 难. 因而缺乏高效的 Petri 网验证工具, 上述提到的研究都只是理论结果, 并未有验证工具. 此外, 也有一些工作针对 Petri 网提出了不同的扩展, 但也有相当高的复杂度[10−14]. 非交互式 Petri 网作为 Petri 网的一个子类, 其验证性质 (可覆盖性、可达性等) 的复杂度是 NP 完备[15].

本文提供了一个异步程序的抽象模型——异步程序标签控制流图 (asynchronous program labelling control flow graph, APG) 系统, 并提出了两种程序验证性质 ϵ 等价性问题和 ϵ 可达性问题, 同时证明这两个问题是 NP 完备的. 我们将 CNF-SAT 规约至这两个问题, 证明其是 NP 难的; 然后, 将这两种性质归约到非交互式 Petri 网的可达性问题, 从而证明它们的 NP 完备性问题. 这两种程序验证性质可以对异步程序的安全性进行近似, 同时验证一些更具体的应用, 如: 动态更新、缺陷定位等. 通过实验室开发的非交互式 Petri 高效工具可以开发出高效的验证工具.

本文第 2 节对用到的预备知识进行概述, 包括 Petri 网、非交互式 Petri 网、上下文无关语法等. 第 3 节给出我们提出的异步程序抽象模型 APG 系统及其语义, 并提出两种验证性质 ϵ 可达性和 ϵ 等价性. 第 4 节引入一种计算模型并行下推系统, 并证明了异步程序的这两种性质是 NP 完备. 第 5 节案例分析, 介绍 APG 如何表示程序以及验证性质如何刻画具体程序问题. 第 6 节总结全文, 并对未来工作进行探讨.

2 预备知识

2.1 基础记号

给定一个有限字母表 $\Sigma=\{v_1,v_2,\ldots,v_k\}$, 符号 Σ^* 和 $\Sigma^{\oplus}$ 分别表示由 Σ 生成的自由幺半群 (free monoid) 和自由交换幺半群 (free commutative monoid). 通俗地说, 即 Σ^* 表示所有由 Σ 中字母组成的字符串, 而 $\Sigma^{\oplus}$ 表示由 Σ 生成的多重集合 (multiset).

记 $\mathbb{N}$ 为自然数集合, $\mathbb{N}^k$ 表示所有定义在自然数上的 k 维向量集合. 定义 Parikh 映射为 $\mathcal{P}:\Sigma^*\to\mathbb{N}^k$. 对每个单词 $w\in\Sigma^*$, 记 $P_w(a_i)$ 为字母 a_i 在单词 w 中的出现次数. 对每个单词 w, 在字母表 Σ 上的 Parikh 映射的象 (Parikh image) 为 $\mathcal{P}(w)=(P_w(a_1),P_w(a_2),\ldots,P_w(a_k))$. Parikh 映射是研究形式语言的一个重要工具. 我们将在后文使用它来辅助证明本文模型中的验证问题.

2.2 Petri 网

定义 1. Petri 网. 一个 Petri 网是一个四元组 (S,T,W,M_0), 其中,

(1) S 和 T 都是不相交的有穷集合, 即 $S\cap T=\varnothing$, S 和 T 分别表示 Petri 网中的库所和迁移, 一般在图中用圆形节点和方形节点表示. Petri 网中的一个对象不会同时是库所和迁移.

(2) $W:(S\times T)\cup(T\times S)\to\mathbb{N}$ 是一个权重函数, 分配给每个弧一个非负的权重.

(3) Petri 网的标记 $M:S\to\mathbb{N}$ 是一组从库所到自然数的映射, 含义是每个库所分配的令牌个数. M_0 表示该 Petri 网的初始标记.

对一个迁移 $t\in T$, 将其前集 (输入库所集) 记为 ${}^\bullet t$, 即 $\{s\in S\,|\,W(s,t)>0\}$, 将其后集 (输出库所集) 记为 $t^\bullet$, 即 $\{s\in S\,|\,W(t,s)>0\}$.

© 中国科学院软件研究所 http://www.jos.org.cn

如果对一个迁移 t 的前集中的任意库所 s, 即 $s \in {}^\bullet t$, 都满足 $M(s) \geqslant W(s,t)$, 那么称这个迁移被标记 M 触发. 迁移 t 被触发后会转移到一个新标记 M_0, 记为 $M \xrightarrow{t} M'$, 其中新标记的令牌数由公式 $M'(s) = M(s) + W(t,s) - W(s,t)$ 确定. 给定一个若干个迁移组成的迁移序列 $T = t_1 t_2 \ldots t_{n-1} t_n$, 两个标记 M_0 和 M', 如果存在标记 $M_1, M_2, \ldots, M_{n-1}$ 使得以下公式成立:

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} M_2 \xrightarrow{t_3} \ldots \xrightarrow{t_{n-1}} M_{n-1} \xrightarrow{t_n} M',$$

则将其记为 $M_0 \xrightarrow{T} M'$, 并称 T 为触发序列, 同时称标记 M'由标记 M_0 可达.

非交互式 Petri 网是一类特殊的 Petri 网, 和另一种描述并发程序的模型基本并行进程 (basic parallel process, BPP) 是等价的.

定义 2. 非交互式 Petri 网. 给定一个 Petri 网 (S,T,W,M_0), 若对于任意 $s \in S, t \in T, |{}^\bullet t| = 1$, 且 $W(s,t) \leqslant 1$, 则称该 Petri 网为非交互式 Petri 网.

图 1 是一个非交互式 Petri 网的例子, 此处每条弧权重设为 1. $S = \{s_1, s_2, s_3, s_4, s_5, s_6\}$表示库所集, $T = \{t_1, t_2, t_3, t_4\}$表示迁移集. 迁移 $t_i (1 \leqslant i \leqslant 4)$ 恰好只有 1 个输入库所, 因此构成非交互式 Petri 网.

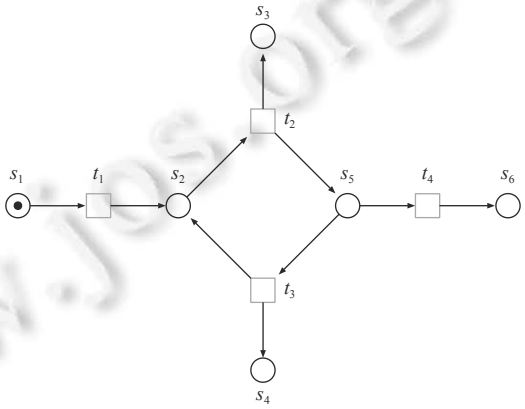

图 1　一个非交互式 Petri 网的例子

问题 1. 非交互式 Petri 的可达性问题. 给定一个非交互式 Petri 网 (S,T,W,M_0), 和该 Petri 的一个标记 M. 则非交互式 Petri 网的可达性问题定义为: 判定标记 M 是否由初始标记 M_0 可达.

针对非交互式 Petri 的可达性问题, 文献 [15] 给出以下结论. 这一结论是本文工作的核心关注点, 利用非交互式 Petri 网的可达性问题的可判定性, 证明后文的相关问题.

定理 1. 文献 [15], Theorem 3.2. 非交互式 Petri 网上可达性问题的复杂度是 NP 完备的.

2.3　上下文无关文法

定义 3. 上下文无关文法. 一个上下文无关文法 (context-free grammer, CFG) 是一个四元组 $G = (Non, Ter, A, P)$, 其中,

- Non 和 Ter 是不相交的有穷集合, 分别表示非终结符集和终结符集.
- $A \in Non$ 表示开始变量, 用来表示整个句子 (或程序).
- $P \subseteq Non \times (Non \bigcup Ter)^*$ 是一个有穷的集合, P 的成员称为文法的规则或产生式.

由上下文无关文法产生的语言称为上下文无关语言. 上下文无关语言的等价性问题是不可判定的. 因此, 我们介绍一类特殊的上下文无关文法: 交换上下文无关文法.

定义 4. 交换上下文无关文法. 一个交换上下文无关语法 (commutative context-free grammer, CCFG) 是一个四元组 $G^c = (Non, Ter, A, P^c)$, 其中 Non 、 Ter 、 A 和前文一致, 而 $P^c \subseteq Non \times (Non \bigcup Ter)^{\oplus}$.

给定两个可交换词 $\alpha, \beta \in (Non \bigcup Ter)^{\oplus}$, 如果由产生式, α 可以直接产生 β, 则记为 $\alpha \to \beta$; 如果经过了若干次

© 中国科学院软件研究所 http://www.jos.org.cn

产生式的使用, α 可以产生 β, 则记为 $\alpha \rightarrow^* \beta$, 其中 $\rightarrow^*$ 表示 $\rightarrow$ 的自反传递闭包.

问题 2. 统一词问题 (uniform word problem). 给定一个交换上下文无关语法 $G^c = (Non, Ter, A, P^c)$, 和一个可交换词 $w \in Ter^{\oplus}$. 统一词问题[16]定义为: 判定 $A \rightarrow^* w$ 是否成立.

定理 2. 文献 [15], Theorem 4.1. 在交换上下文无关文法中, 统一词问题的复杂度是 NP 完备的.

证明的思路是将统一词问题归约到非交互式 Petri 网的可达性问题, 即将上下文无关文法翻译为非交互式 Petri 网, 那么可交换词 w 为该 Petri 网中的一个可达标记当且仅当 $A \rightarrow^* w$ 成立. 也就是说该文献表明, 对于每个交换上下文无关文法 $\mathcal{G}$, 都存在一个非交互式 Petri 网 $\mathcal{V}$, 使得对 $w \in \mathcal{L}_{\mathcal{G}}(N)$ 的成员资格查询可以通过对 $\mathcal{V}$ 的可达性查询来回答.

3 程序模型

本节提出异步程序标签控制流图 (APG) 系统, 以控制流分析方法来辅助刻画异步程序的特征. 我们首先简要介绍一下如何使用控制流图来形式化表示异步程序.

3.1 APG 系统

定义 5. APG 系统. APG 系统表示异步程序标签控制流图 G (asynchronous program labelling control flow graph). G 可以用一个元组 (P, E, V) 来表示, 其中,

(1) $P = \{p_1, p_2, \ldots, p_n\}$ 是一个有穷的过程 (procedures) 名称集, 用于表示可以被调用的过程集合.

(2) V 是一个控制节点集, 每个控制节点代表程序的一个基本块 (basic block), 即对应一段异步程序中最大顺序执行的代码序列.

(3) $E \subseteq V \times L \times V$ 是一个边集, 其中 L 是一个标签集合, E 表示节点到节点的迁移或跳转关系.

对于每个过程 $p \in P, p$ 在控制节点集 V 中都有一个入口节点 v_p^s (start) 和一个出口节点 v_p^e (end), 分别表示程序在这过程 p 执行开始和结束的位置. 过程 p 可以在任何时候被调用.

需要注意的是, 我们定义的 APG 系统是良结构的. 我们用 V_p 表示控制节点集 V 中属于过程 p 的控制节点集合, 则有 $V_p \subseteq V$. 良结构的意思是说, V_p 中任意节点都能从入口节点 v_p^s 出发可达, 并且可到达出口节点 v_p^e.

而标签集合 L 中的标签共有以下 3 种类型.

(1) $u \xrightarrow{\epsilon} v \in E$, ϵ 标签为空标签, 表示不存在过程调用, 仅存在一些状态的迁移. 控制流在过程内部执行, 从控制节点 u 直接转移到控制节点 v.

(2) $u \xrightarrow{p^s} v \in E$, p^s 标签为同步调用标签, 表示控制节点 u 同步 (synchronous) 调用了过程 p, 并且在过程 p 调用完成后控制流返回到控制节点 v.

(3) $u \xrightarrow{p^a} v \in E$, p^a 标签为异步调用标签, 表示控制节点 u 异步 (asynchronous) 调用了过程 p, 无需等待过程 p 返回, 控制流直接转移到控制节点 v.

3.2 APG 系统的语义

我们在 APG 系统中引入数据流分析方法, 来分析异步程序的行为. 令 D 表示一个数据流值的有限集合, T 表示一个数据流转移函数, 即 $T : E \rightarrow D \rightarrow D$, T 将 APG 系统中边集一条边 e 和一个数据流值 d 映射到另一个数据流值 d' 上.

现在我们给出 APG 系统的格局定义.

定义 6. APG 系统的格局. APG 系统的格局用一个三元组 (Pr, M, N) 来表示, 其中,

(1) $Pr \in ((V \times D) \times V^*)^{\oplus}$ 是一个带有程序堆栈的抽象状态的多重集, 抽象状态指控制节点和数据流值, 栈符号则表示控制节点.

(2) $M \in P^{\oplus}$, 其中 M 表示处于 pending 状态的异步调用, 即还在等待执行的调用.

(3) $N \in P^{\oplus}$, 其中 N 表示处于 returned 状态的异步调用, 即执行结束后已经返回的调用.

在引入以上格局的记号后, 若用记号 *main* 表示程序入口, 则对一个异步程序来说, 其对应的 APG 系统的初始

© 中国科学院软件研究所 http://www.jos.org.cn

格局集合为:

$$\left(\left\{\left(\left(v^s_{main},d^0_{main}\right),\epsilon\right)\right\},\varnothing,\varnothing\right).$$

此时, 程序在过程 *main* 入口节点 v^s_{main} 开始执行, 初始数据流值为 d^0_{main}, ϵ 表示程序栈为空, pending 和 returned 调用也都为空集 $\varnothing$.

定义 7. APG 系统的语义. 针对一个异步程序的行为, 我们将其分为以下几类, 并定义 APG 系统的语义如下.

(1) 过程内部操作:

$$\frac{e=v\xrightarrow{\epsilon}v'}{(\{((v,d),\sigma)\}\uplus Pr,M,N)\to(\{((v',T(e)(d)),\sigma)\}\uplus Pr,M,N)}.$$

此时程序的全局抽象状态被改变, 由 (v,d) 变为 $(v',T(e)(d))$. 由于只是过程的内部操作, 栈顶符号仍为 σ, 保持不变; M 和 N 也保持不变.

(2) 同步调用:

$$\frac{e=v\xrightarrow{p^s}v'}{(\{((v,d),\sigma)\}\uplus Pr,M,N)\to\left(\left\{\left(\left(v^s_p,T(e)(d)\right),\sigma\cdot v'\right)\right\}\uplus Pr,M,N\right)}.$$

此时程序的全局抽象状态被改变, 控制节点变为同步调用入口节点 v^s_p, 数据流值由 d 通过转移函数变为 $T(e)(d)$, 栈顶符号发生变化, 将 v' 压入栈 σ, 变为 $\sigma\cdot v'$; 由于是同步调用, 因此 M 和 N 也不会发生变化.

(3) 同步返回:

$$\frac{}{\left(\left\{\left(\left(v^e_p,d\right),\sigma\cdot v'\right)\right\}\uplus Pr,M,N\right)\to(\{((v',T(e)(d)),\sigma)\}\uplus Pr,M,N)}.$$

此时程序的全局抽象状态被改变, 控制节点从出口节点 v^e_p 返回到 v', 栈顶符号也发生变化, 此时需要将 v' 从栈顶弹出, 栈顶符号变为 σ.

(4) 异步 post:

$$\frac{e=v\xrightarrow{p^a}v'}{(\{((v,d),\sigma)\}\uplus Pr,M,N)\to(\{((v',T(e)(d)),\sigma)\}\uplus Pr,\{p\}\uplus M,N)}.$$

异步调用需要发送一个 post 请求, 与同步调用类似, 但异步调用不需要等待调用返回, 而是让调用进入 pending 状态, 所以控制节点不会变为入口节点 v^s_p, 而是直接变为 v', 数据流转移则于同步调用类似, 栈顶元素不会发生变化; M 中则加入一个处于 pending 状态的异步调用 p, N 则不变.

(5) 异步调用 dispatch:

$$\frac{}{(Pr,\{p\}\uplus M,N)\to\left(\left\{\left(\left(v^s_p,d^0_p\right),\epsilon\right)\right\}\uplus Pr,M,N\right)}.$$

异步调用中处于 pending 状态的调用 p 总会被某个线程执行, 此时表现为异步调用分发 (dispatch). 程序全局状态发生改变, 控制节点变为入口节点 v^s_p, 而 M 则需要减少一个正在 pending 状态的调用 p, 由一开始的 $\{p\}\uplus M$ 变为 M.

(6) 异步返回:

$$\frac{}{\left(\left\{\left(\left(v^e_p,d\right),\epsilon\right)\right\}\uplus Pr,M,N\right)\to(Pr,M,\{p\}\uplus N)}.$$

异步返回表示该调用处于 returned 状态, 控制节点从出口节点 v^e_p 离开, 程序全局抽象状态回到 Pr; M 不会改变, N 则加入一个处于 pending 状态的异步调用 p.

至此, 我们在 APG 系统上定义了异步程序的行为.

3.3 安全性性质和验证问题

真实世界中大多数实际运行的程序, 都会由于语法、语义的出错导致出现与预期不符的异常行为. 而异步程序由于其涉及多线程间通讯等并发要求, 编写起来相当困难, 也容易出现异常行为, 更难以进行软件测试. 解决方案是通过形式化的方法, 将程序转化为程序模型, 使用模型检测等验证技术, 找出程序模型的错误以及异常行为, 可以从本质上为我们完成对程序的验证, 以此保证程序的安全性.

© 中国科学院软件研究所 http://www.jos.org.cn

异步程序的可达性问题以及等价性问题都是不可判定的. 因此本文通过对模型作出一些限制, 重点讨论以下两个问题: 异步程序中的ϵ可达性问题以及ϵ等价性问题.

而借助非交互式 Petri 网的可判定性以及相关高效的验证工具, 可以来对程序的安全性性质进行验证. 于是我们引入 APG 系统的ϵ可达性问题与ϵ等价性问题, 以及第 4 节提及的并行下推系统, 并将其最终归约至非交互式 Petri 网的可达性上.

定义 8. APG 系统的ϵ可达性问题. 给定一个 APG 系统$G=(P,E,V)$, 一个格局(M,N), 在G中所有从初始$init$状态经过若干次迁移能到达的格局集合称为ϵ-*reachable* 集合. 记迁移规则$\rightarrow$的自反传递闭包为$\rightarrow^*$. 则ϵ-*reachable* 集合可以写为$Reach_\epsilon(G)=\{(M,N)|init\rightarrow^*(\varnothing,M,N)\}$.

APG 系统的ϵ可达性问题定义为: 给定一个格局(M,N), 判断格局(M,N)是否在$Reach_\epsilon(G)$中. 若格局(M,N)在$Reach_\epsilon(G)$中, 则称格局(M,N)可达. 程序的ϵ可达性问题又被称为安全性问题, 这是因为往往可以通过验证程序错误或异常状态的不可达来说明程序的安全性.

定义 9. APG 系统的ϵ等价性问题. 给定两个异步程序, 其所对应的 APG 系统为$G_1=(P_1,E_1,V_1),G_2=(P_2,E_2,V_2)$.

APG 系统的ϵ等价性问题定义为: 判定这两个程序的ϵ可达集合是否相等, 即$Reach_\epsilon(G_1)$是否等于$Reach_\epsilon(G_2)$, 若二者相等, 则这两个程序等价. ϵ等价性问题又称为ϵ安全等价性问题, 例如, 给定一个异步程序和一个截止日期, 询问它是否可以在每种可能性上准时执行.

对于异步程序中的ϵ可达性问题以及ϵ等价性问题来说, 我们讨论其复杂度下界. 因此我们给出以下引理.

引理 1. 异步程序的ϵ可达性问题的复杂度是 NP-hard 的.

引理 2. 异步程序的ϵ等价性问题的复杂度是 NP-hard 的.

对于引理 1 和引理 2, 我们并不直接给出证明. 我们将在第 4 节中证明更强的结论, 即异步程序上的ϵ可达性问题和ϵ等价性问题的复杂度是 NP 完备的. 证明的主要思路是通过引入并行下推系统, 使用归约手段, 将 3CNF-SAT 问题归约到并行下推系统的这两个问题.

在本节的最后, 我们用一个较为简单的例子来说明如何使用 APG 系统来对异步程序进行验证. 图 2 表示一个异步程序代码片段, 用来模拟向客户端发送请求处理任务. *r* 是一个*request_list*类型的全局指针, 用于表示请求列表中的当前请求. 程序入口在*main*, 这段程序首先初始化 *r*, 然后异步调用函数*reqs*, 随后开始循环调度. 函数*reqs*用来向远程客户端发送请求. 其逻辑是, 如果请求列表 *r* 为空, 则异步调用其自身; 否则为变量*rc* (表示远程客户端) 分配一块内存, 若无内存可分配, 返回一个超出内存限制异常, 否则异步调用函数*client*, 随后执行下一个请求列表中的下一个请求. 函数*client*用于处理请求相关任务.

```
1.  global request_list *r;
2.  main(){
3.     ... //setup request list r
4.     async reqs();
5.     ...//dispatch loop
6.  }
7.
8.  reqs(){
9.     if(r==NULL){
10.       async reqs();
11.       return;
12.    }
13.    rc = malloc(...);
14.    if(rc==NULL){
15.       return ABORT_OUT_OF_MEM;
16.    }
17.    async client(rc,r->id);
18.    r = r->next;
19.    reqs();
20. }
21. client(client_t *c, int id){
22.    ...//setup
23.    c->id = id;
24.    ...//continue processing
25.    return;
26. }
```

图 2　一个异步程序的代码片段

对于图 2 中的异步程序, 我们可以将过程调用抽象成集合$P=\{p_{main},p_{reqs},p_{client}\}$, 程序中最大顺序执行的代码序列抽象成控制节点集 *V*, 控制流的迁移关系抽象为边集 *E*. 对 *E* 中边上的标签进行分类. 针对调用的类型以及是否调用, 将其分为异步调用和同步调用以及过程内不涉及调用, 并将这 3 种类型抽象为控制流图中的 3 种不同标签. 最后通过以上方式得到一个异步程序标签控制流图 (APG), 来刻画这个异步程序.

在这个异步程序的 APG 系统中, 每个 procedure 对应的控制流图如图 3 所示. 函数*main*为入口, 因此在此省

© 中国科学院软件研究所 http://www.jos.org.cn

略其控制流图. APG 系统的可达性可以由其对应的并行下推系统归约至非交互式 Petri 网来回答, 因此可以验证程序的安全性等性质.

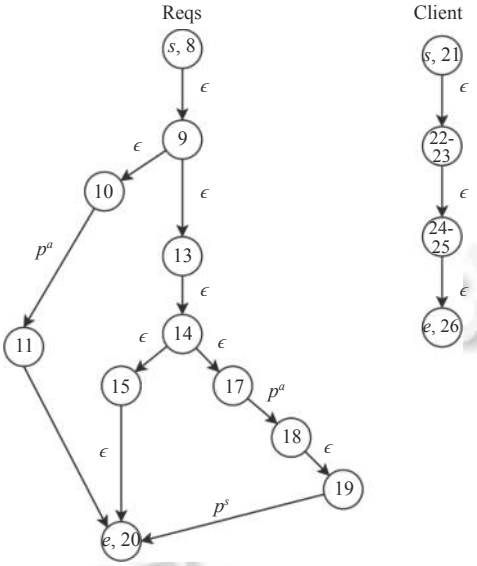

图 3 图 2 中所有 procedure 对应的控制流图

4 并行下推系统

我们引入计算模型并行下推系统 (parallel pushdown system, PPS), 将 APG 系统上的 ϵ 可达性与 ϵ 等价性转为对应 PPS 上的验证性质. 并行下推系统的语义可以看成是多个下推系统无通讯地并行执行, 即:

$$\text{PPS} = \text{PDS} \| \text{PDS} \| \ldots \| \text{PDS}.$$

4.1 并行下推系统的语义

为了转化 APG 系统上的 ϵ 可达性与 ϵ 等价性, 我们首先给出并行下推系统的格局定义以及其语义.

定义 10. 并行下推系统的格局. 并行下推系统的格局用一个三元组 (Pr, M, N) 来表示, 其中 Pr, M, N 与前文 APG 系统保持一致.

定义 11. 并行下推系统的语义. 我们可以通过重新定义全局抽象状态集合, 并忽略 APG 系统中同步调用的标签, 将一个程序所对应的 APG 系统 $(G = (P, V, E), main)$ 转化成一个并行下推系统 PPS_G. $PPS_G = \left(Q, q^0, Q^f, P_\epsilon, \Gamma_\perp, \delta, init\right)$, 其中,

(1) $Q = (V \times D) \cup \{q_p^0\}_{p \in P} \cup \{q_p^f\}_{p \in P}$ 是状态的有穷集合.

(2) $q^0 : P \to Q$, 对每个 p, 有 $q^0(p) = q_p^0$, 表示起始状态.

(3) $Q^f : P \to \mathcal{P}(Q)$, 对每个 p, 有 $Q^f(p) = \left\{q_p^f\right\}$, 表示最终被接收的状态的集合.

(4) $P_\epsilon = P \cup \{\epsilon\}$ 是输入字母表集合.

(5) $\Gamma_\perp = \{v | v' \xrightarrow{p^s} v \in E \text{ for some } p \in P\} \uplus \{\perp\}$, 表示栈字符集.

(6) *init* 表示并行下推系统的初始格局.

转移函数 $\delta \subseteq (Q \times \Gamma_\perp) \times P_\epsilon \times (Q \times \Gamma^*)$ 是满足以下条件的最小集合.

(1) 令 $e = v \xrightarrow{\epsilon} v' \in E$, 对每个 $u \in \Gamma_\perp$ 和 $d \in D$, 都有以下公式成立:

$$((v, d), u) \xrightarrow{\epsilon} ((v', T(e)(d)), u) \in \delta.$$

(2) 令 $e = v \xrightarrow{p^s} v' \in E$, 对每个 $u \in \Gamma$ 和 $d \in D$, 都有以下公式成立:

© 中国科学院软件研究所 http://www.jos.org.cn

$$((v,d),u) \xrightarrow{\epsilon} \left(\left(v_p^s, T(e)(d)\right), u\cdot v'\right) \in \delta,$$

$$\left(\left(v_p^e, d\right), v'\right) \xrightarrow{\epsilon} ((v',d),\epsilon) \in \delta.$$

(3) 令 $e = v \xrightarrow{p^a} v' \in E$ 表示 APG 中一条异步调用的边, 对每个 $u \in \Gamma$ 和 $d \in D$, 都有以下公式成立:

$$((v,d),u) \xrightarrow{p} ((v',T(e)(d)),u) \in \delta.$$

(4) 对每个 $p \in P$, 都有以下公式成立:

$$\left(q_p^0, \epsilon\right) \xrightarrow{\epsilon} \left(\left(v_p^s, d_p^0\right), \perp\right) \in \delta.$$

(5) 对每个 $p \in P$ 和每个 $d \in D$, 都有以下公式成立:

$$\left(\left(v_p^e, d\right), \perp\right) \xrightarrow{\epsilon} \left(q_p^f, \epsilon\right) \in \delta.$$

注意到, 标签是表示异步调用的过程名称, 并且栈字符集保存的是同步调用返回后的控制节点. 并行下推系统 PPS_G 的迁移规则 δ 是继承其对应的 APG 系统的规则, 并通过将同步调用 p^s 标签视为 ϵ 标签后得到的. 因此从 APG 系统转化而来的并行下推系统也可以视为空栈限制, 即其格局中的 Pr 为 $\varnothing$.

4.2 并行下推系统 ϵ 可达性问题的复杂度下界

异步程序的 ϵ 可达性问题被转化为在对应的 PPS_G 系统的 ϵ 可达性问题. 与 APG 系统 ϵ 可达性问题的定义相似, 我们给出其对应并行下推系统 ϵ 可达性问题的定义.

定义 12. 并行下推系统的 ϵ 可达性问题. 给定一个格局 (M,N), 并行下推系统的 ϵ 可达性问题定义为: 判断格局 (M,N) 是否由初始格局 $init$ 可达.

引理 3. 并行下推系统 (PPS) 上 ϵ 可达性问题的复杂度是 NP 难的.

证明: 我们从 3-CNF-SAT 问题可以归约到并行下推系统的 ϵ 可达性问题来证明引理 3.

给定一个 3-CNF 公式 $\varphi = \bigwedge c_i$, $C = \{c_1, c_2, \ldots, c_m\}$ 是定义在有限布尔变量集 $U = \{u_1, u_2, \ldots, u_n\}$ 上所有子句的集合. 我们将过程 p_{u_i} 构造为文字 (literal), 将过程 p_{c_j} 构造为子句 (clause).

过程 p_{u_i} 有两条从起始节点出发的出边, 分别代表 $u_i = \text{true}$ 和 $u_i = \text{false}$.

首先考虑 $u_i = \text{true}$ 的情况. 令 v_T 和 v_F 分别为代表 $u_i = \text{true}$ 和 $u_i = \text{false}$ 的节点. 令 C_i 是所有包含 u_i 的字句的集合. 那么对于 u_i 变量对应的过程 p_{u_i}, 我们将其行为定义为: 从 v_T 出发, 过程 p_{u_i} 顺序地异步调用变量 $c_j \in C_i$ 对应的过程 p_{c_j}, 然后转到出口节点.

对于 $u_i = \text{false}$ 的情况可以类似地定义. 考虑变量 c_j 对应的过程 p_{c_j}, 将其行为定义为它异步调用自身到起始节点或者直接转到出口节点的过程. p_{main} 顺序地同步调用进程 p_{u_i} $(1 \leqslant i \leqslant m)$, 然后转到出口节点.

现在我们考虑格局 (M,N) 的 ϵ 可达性问题, 此时有 $M = \uplus_{i=1}^{n} \{p_{c_1}, p_{c_2}, \ldots, p_{c_m}\}$, 并且 $N = \{p_{main}\}$.

因此根据以上构造方法, 我们可以得到格局 (M,N) 是 ϵ 可达的当且仅当公式 φ 是可满足的. 并且构造在多项式时间内可以完成.

综上, 引理 3 得证.

4.3 安全性性质和验证问题

异步程序的 ϵ 等价性问题被转化为在对应的 PPS_G 系统的 ϵ 交换上下文无关文法等价问题.

对于 PPS_G 系统来说, 其可识别的单词由其中下推系统接受的单词合并得到. 而在 PPS_G 系统的上下文语义中, 单词指的是异步调用的序列.

引理 4. 给定一个并行下推系统 PPS_G, 一个格局 (M,N). 则格局 (M,N) 由初始格局可达 (即 $init \xrightarrow{w} (\varnothing, M, N)$) 当且仅当存在单词 $w_{p_i}^1, w_{p_i}^2, \ldots, w_{p_i}^{n_i}$ 被初始状态为 $q_{p_i}^0$ 和最终状态为 $q_{p_i}^f$ 的下推自动机 $(1 \leqslant i \leqslant |P|)$ 所接受, 且满足以下条件.

- w 由所有 $w_{p_i}^j$ 合并得到, 其中 $1 \leqslant j \leqslant n_i, 1 \leqslant i \leqslant |P|$.
- 对 $p_i = main$, 有 $P_w(p_i) = m_i + n_i - 1$, 对 $p_i \neq main$, 有 $P_w(p_i) = m_i + n_i$.

© 中国科学院软件研究所 http://www.jos.org.cn

• 对于任意前缀 $w'=u_1u_2\ldots u_{k'}$ 以及 p_i:

① 如果 $p_i\neq main$, 则有 $|\{j|u_l \twoheadleftarrow w_{p_i}^j, 1\leqslant l\leqslant k'\}|<P_{w'}(p_i)$.

② 如果 $p_i=main$, 则有 $|\{j|u_l \twoheadleftarrow w_{p_i}^j, 1\leqslant l\leqslant k'\}|<P_{w'}(p_i)+1$.

证明: 我们只需要证明对于并行下推系统中的 ϵ 可达性问题, 存在等价的多个初始状态为 $q_{p_i}^0$ 和最终状态为 $q_{p_i}^f$ 的下推自动机 $(1\leqslant i\leqslant |P|)$ 与它对应.

"$\Longrightarrow$ 方向". 由于并行下推系统 PPS_G 是由多个下推系统无通讯地并行执行, 因此对每个过程 $p\in P$, 如果格局 (M,N) 由并行下推系统的初始格局可达, 即有 $init\xrightarrow{w}(\varnothing,M,N)$ 成立, 说明存在若干个迁移 w_i, 使得 $init\to^*(\varnothing,M,N)$. 不妨设 $w=w_1w_2\ldots w_P$, 其中 $w_i\in\Gamma^*$. 则对 $w_i(1\leqslant i\leqslant |P|)$, 令 $w_i=w_{p_i}^1,w_{p_i}^2,\ldots,w_{p_i}^{n_i}$, 则我们可为其构造一个等价的初始状态为 $q_{p_i}^0$ 和最终状态为 $q_{p_i}^f$ 的下推自动机 M_i, 使得单词 w_i 被 M_i 识别. 如果 p_i 是入口 $main$, 则不会被其他过程所调用, 因此有 Parikh 映射的象 $P_w(p_i)=(m_i-1)+n_i=m_i+n_i-1$ 成立; 若 p_i 不是入口 $main$, 则有 $P_w(p_i)=m_i+n_i$. 同理对任意前缀 $w'=u_1u_2\ldots u_{k'}$ 以及 p_i, 若 $p_i=main$, 则有 $P_{w'}(p_i)=m_i'+n_i'-1>m_i'-1\geqslant|\{j|u_l \twoheadleftarrow w_{p_i}^j, 1\leqslant l\leqslant k'\}|-1$, 反之亦有 $P_{w'}(p_i)\geqslant|\{j|u_l \twoheadleftarrow w_{p_i}^j, 1\leqslant l\leqslant k'\}|$.

"$\Longleftarrow$ 方向". 如果存在单词 $w_{p_i}^1,w_{p_i}^2,\ldots,w_{p_i}^{n_i}$ 被初始状态为 $q_{p_i}^0$ 和最终状态为 $q_{p_i}^f$ 的下推自动机 M_i 所识别, 其中 $M_i=\left(Q,\Sigma,\Gamma,\delta,q_{p_i}^0,z_0,\left\{q_{p_i}^f\right\}\right),(1\leqslant i\leqslant|P|)$. 那么只需要证明此时并行下推系统的格局 (M,N) 由 $init$ 可达. 将所有单词合并后得到 w. 这个 w 即表示异步调用的序列, 一个迁移序列使得 $init\xrightarrow{w}(\varnothing,M,N)$ 成立, 即并行下推系统中的格局 (M,N) 可达.

综上, 引理 4 得证.

引理 5. 存在一个多项式的归约方法, 使得 PPS_G 上的 ϵ 可达性问题可以归约到交换上下文无关进程的可达性问题.

证明: 由引理 4, PPS_G 中的 ϵ 可达性问题可以等价于存在单词 $w_{p_i}^1,w_{p_i}^2,\ldots,w_{p_i}^{n_i}$ 被初始状态为 $q_{p_i}^0$ 和最终状态为 $q_{p_i}^f$ 的自动机 M_i 所接受, 其中 $1\leqslant i\leqslant|P|$.

因此, 我们只需要证明可以将 PPS_G 中的下推自动机 M_i 与一个交互上下文无关文法等价. 而每一个下推自动机与一个上下文无关文法等价. 根据定义 4, 只需要证明其对应的上下文无关文法中的产生式右边的符号与顺序无关, 是一个多重集.

因此 PPS_G 上的 ϵ 可达性问题可以归约到交换上下文无关进程的可达性问题.

综上, 引理 5 得证.

定理 3. 异步程序的 ϵ 可达性问题的复杂度是 NP 完备的.

证明: 由于异步程序的 ϵ 可达性问题被转化为在对应的 PPS_G 系统的 ϵ 可达性问题. 而由引理 5, 存在一个多项式时间的归约方法, 使得 PPS_G 上的 ϵ 可达性问题可以归约到交换上下文无关进程的可达性问题. 又因为交换上下文无关文法和非交互式 Petri 网等价, 因此异步程序的 ϵ 可达性问题是 NP 完备的.

综上, 定理 3 得证.

对异步程序的安全等价性问题来说, 我们考虑其 ϵ 可达集. 非交互式 Petri 网的语言等价性是不可判定的, 可达集等价问题也是困难的.

我们使用形式语言理论中的 Parikh 引理[17]来证明我们关于异步程序 ϵ 等价性问题结论. Parikh 引理表述如下.

引理 6. Parikh 引理. 对任意上下文无关语言 $\mathcal{L}$, 存在一个高效的可计算正则语言 $\mathcal{L}'$, 满足: $\text{Parikh}(\mathcal{L})=\text{Parikh}(\mathcal{L}')$.

通常来说, 这个正则语言的构建需要指数时间 EXPTIME. 但是在这个问题中, 由于我们考虑的 ϵ 标签表示的是程序的同步调用, 因此我们有比 Parikh 引理更简洁的表示.

定理 4. 异步程序的 ϵ 等价性问题的复杂度是 NP 完备的.

证明: 异步程序的 ϵ 等价性问题可以转化为求解其对应的 ϵ 可达集的等价性. 又由引理 5 可知, 其 ϵ 可达集是有上下文无关文法产生的上下文无关语言. 因此根据 Parikh 引理, 存在一个高效的可计算正则语言, 使得二者的 Parikh 映射的象相同. 由于我们考虑将同步调用也视为 ϵ 标签, 因此构建这个正则语言的复杂度可以进一步降为 NP 完备.

© 中国科学院软件研究所 http://www.jos.org.cn

综上, 定理 4 得证.

5 案例分析

本节以一个具体实例来说明如何借助异步程序 ϵ 可达性与 ϵ 等价性来验证程序安全性与活性, 甚至动态更新、错误定位等性质.

图 4 是一个带有 bug 的 server 程序例子. 这个程序执行一个循环调度 server 函数, 并等待外部客户端进行连接, 且以异步方式读取数据. 如果有一个客户端连接上了服务器, server 函数会分配一个 client 数据结构, 以异步 post 方式处理 client 的连接 (process_client 函数), 在结束后调用自身以等待下一次连接. process_client 函数根据 client 的状态完成数据的读取或发送, 如果为 TO_READ 则执行 read 函数, DONE_READ 则执行 send 函数. 如果发生某些错误 (第 14 行), read 函数可以将连接断开, 否则就读取数据. 如果 read 函数读取数据时, client 的数据还没有完全读取, 则将其赋为 TO_READ 状态, 若完全读取, 则赋为 DONE_READ 状态. 在 read 函数执行到最后时, 会以同步方式调用 process_client 函数. send 函数则仅当 client 状态为 DONE_READ 时被调用, 最后调用 disconnect 函数, 将状态改为 CLOSED.

```
1.  main(){                          26. send(c){
2.      server();                    27.     assert(c->state== DONE_READ);
3.  }                                28.     disconnect(c);
4.                                   29. }
5.  server(){                        30.
6.      client *c = malloc(...);     31. process_client(c){
7.      if(c != NULL){               32.     if(c->state == TO_READ){
8.          c->state = TO_READ;      33.         async read(c);
9.          async process_client(c); 34.         return;
10.     }                            35.     }
11.     async server;                36.     if(c->state == DONE_READ){
12. }                                37.         async send();
13.                                  38.         return;
14. read(c){                         39.     }
15.     if(*){                       40.     assert(false);
16.         disconnect(c);//BUG      41. }
17.     }else{                       42.
18.         if(*){                   43. disconnect(c){
19.             c->state = TO_READ;  44.     c->state = CLOSED;
20.         }else{                   45.     return;
21.             c->state= DONE_READ; 46. }
22.         }
23.     }
24.     process_client(c);
25. }
```

图 4 一个带有 bug 的 server 程序例子

这个程序的正确行为是通过断言 (assert 函数) 的方式确保 *client* 的状态处于正常. 例如在 send 函数的 assert 肯定会满足, 这是因为已经在第 36 行 process_client 函数中检查过. 然而, 这个例子在第 15 行有一个 bug, 在 disconnect 函数执行结束后, client 的状态被设为 CLOSED, 此时程序由第 15 行跳到第 24 行执行, 而在 process_client 函数中, 并没有针对 CLOSED 状态的检查, 因此程序出现了未知行为.

针对这一情况, 我们可以借助 ϵ 可达性来验证这个异步程序的断言是否正确. 核心思路是通过说明坏状态的不可达来确保程序的安全. 在这个例子中, 我们可以为这个异步程序构建一个 APG 系统. 图 5 表示了这个异步程序的 APG 系统中所有 procedure 对应的控制流图.

图 5 中每个控制流图有一个入口节点 s 与出口节点 e, 每条边上带有对应的标签 ϵ 或 p^a 或 p^s. 注意每个节点中的内容为代码的行号, 表示其对应的一个基本块. 而 p 则表示程序某个 procedure 的名称, 例如 server 对应的控制流图中, 节点 7-8 到节点 10 上的 p 指 process_client 函数, 而节点 10 到节点 11 上的 p 指其自身 server 函数.

© 中国科学院软件研究所 http://www.jos.org.cn

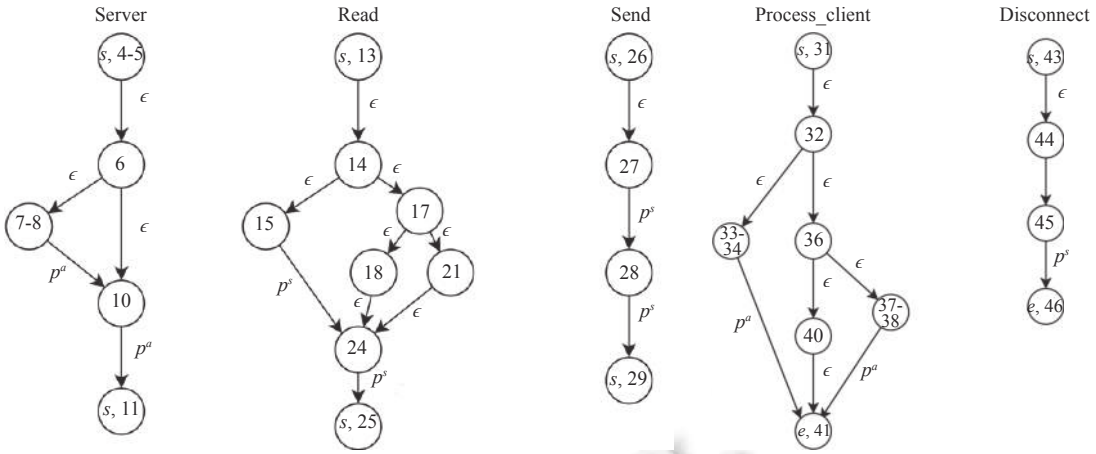

图 5 APG 系统中所有 procedure 对应的控制流图

对于异步程序的安全性性质, 我们可以先找到表示坏状态的格局 (M', N'), 其中 M' 为程序从第 15 行执行到第 24 行时还处于 pending 状态的调用, N' 表示那时已经返回的调用. 我们可以借助其对应并行下推系统的可达性来回答 (M', N') 是否在 APG 系统中格局可达. 在这个例子中, (M', N') 是可达的, 因此说明程序会到达一个坏状态, 验证了其安全性.

我们提出模型上的两个问题可以验证一些异步程序性质, 比如: 我们可以借助 ϵ 等价性来判断程序在动态更新前后其 ϵ 可达集是否变化. 在这个例子中, 如果原程序运行时一个动态补丁发生效果, 例如在第 15 行与第 16 行之间加入对客户端状态的断言以及 return 语句. 则如果其 ϵ 可达集在动态更新之后, 与原程序的可达集不等价, 那么说明程序的安全性发生变化, 即完成了对程序动态更新的安全性验证. 这说明我们可以使用 ϵ 等价性来判断一个程序是否被正确的动态更新.

6 总结及未来工作

本文关注基于非交互式 Petri 网的异步程序验证, 提出了用于刻画异步程序模型 APG 系统, 引入计算模型并行下推系统 PPS_G, 将异步程序的 ϵ 可达性问题和 ϵ 等价性问题被转化为在对应的 PPS_G 上的可达性问题与交换上下文无关文法的等价性问题. 同时, 证明了异步程序上的这两个性质即 ϵ 可达性和 ϵ 等价性都是 NP 完备的.

未来的工作包括以下两方面.

• 在实现方面, 我们希望通过借助非交互式 Petri 网的验证工具[18], 来得到一个用于自动化验证 APG 系统的验证器. 并且针对 Java 程序做出一个异步程序的自动化验证工具, 可以对异步程序的缺陷定位、动态更新等进行验证.

• 在理论方面, 我们希望引入不同线程之间的通讯, 从而得到异步通讯程序在堆栈和通信近似下的更易于处理的模型. 并且希望利用下推自动机、非交互 Petri 网的相关特点, 将其结合后得到更有效的验证问题, 以此验证异步通讯程序.

References:

[1] Ramalingam G. Context-sensitive synchronization-sensitive analysis is undecidable. ACM Trans. on Programming Languages and Systems, 2000, 22(2): 416–430. [doi: 10.1145/349214.349241]

[2] Leroux J, Praveen M, Sutre G. Hyper-ackermannian bounds for pushdown vector addition systems. In: Proc. of the Joint Meeting of the 23rd EACSL Annual Conf. on Computer Science Logic (CSL) and the 29th Annual ACM/IEEE Symp. on Logic in Computer Science (LICS). Vienna: ACM, 2014. 1–10. [doi: 10.1145/2603088.2603146]

[3] Leroux J, Sutre G, Totzke P. On the coverability problem for pushdown vector addition systems in one dimension. In: Proc. of the 42nd Int'l Colloquium on Automata, Languages, and Programming. Kyoto: Springer, 2015. 324–336. [doi: 10.1007/978-3-662-47666-6_26]

[4] D'Osualdo E, Kochems J, Ong CHL. Automatic verification of Erlang-style concurrency. In: Proc. of the 20th Int'l Static Analysis Symp.

© 中国科学院软件研究所 http://www.jos.org.cn

Seattle: Springer, 2013. 454–476. [doi: 10.1007/978-3-642-38856-9_24]

[5] Sen K, Viswanathan M. Model checking multithreaded programs with asynchronous atomic methods. In: Proc. of the 18th Int'l Conf. on Computer Aided Verification. Seattle: Springer, 2006. 300–314. [doi: 10.1007/11817963_29]

[6] Petri CA. Kommunikation mit automaten [Ph.D. Thesis]. Bonn: University of Bonn, 1962.

[7] Ganty P, Majumdar R. Algorithmic verification of asynchronous programs. ACM Trans. on Programming Languages and Systems, 2012, 34(1): 6. [doi: 10.1145/2160910.2160915]

[8] Atig MF, Bouajjani A, Kumar KN, Saivasan P. Verification of asynchronous programs with nested locks. In: Proc. of the 37th IARCS Annual Conf. on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017). Dagstuhl: Schloss Dagstuhl-Leibniz-Zentrum Fuer Informatik, 2018. 11.

[9] Emmi M, Ganty P, Majumdar R, Rosa-Velardo F. Analysis of asynchronous programs with event-based synchronization. In: Proc. of the 24th European Symp. on Programming Languages and Systems. London: Springer, 2015. 535–559. [doi: 10.1007/978-3-662-46669-8_22]

[10] Lakos C. The consistent use of names and polymorphism in the definition of object Petri nets. In: Proc. of the 17th Int'l Conf. on Application and Theory of Petri Nets. Osaka: Springer, 1996. 380–399. [doi: 10.1007/3-540-61363-3_21]

[11] Valk R. Petri nets as token objects. In: Proc. of the 19th Int'l Conf. on Application and Theory of Petri Nets. Lisbon: Springer, 1998. 1–24. [doi: 10.1007/3-540-69108-1_1]

[12] Christensen S, Hansen ND. Coloured Petri nets extended with channels for synchronous communication. In: Proc. of the 15th Int'l Conf. on Application and Theory of Petri Nets. Zaragoza: Springer, 1994. 159–178. [doi: 10.1007/3-540-58152-9_10]

[13] Czerwiński W, Fröschle S, Lasota S. Partially-commutative context-free processes. In: Proc. of the 20th Int'l Conf. on Concurrency Theory. Bologna: Springer, 2009. 259–273. [doi: 10.1007/978-3-642-04081-8_18]

[14] Yang QZ, Li GQ. Model on asynchronous communication program verification based on communicating Petri nets. Ruan Jian Xue Bao/Journal of Software, 2017, 28(4): 804−818 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5191.htm [doi: 10.13328/j.cnki.jos.005191]

[15] Esparza J. Petri nets, commutative context-free grammars, and basic parallel processes. Fundamenta Informaticae, 1997, 31(1): 13–25. [doi: 10.3233/FI-1997-3112]

[16] Huynh DT. Commutative grammars: The complexity of uniform word problems. Information and Control, 1983, 57(1): 21–39. [doi: 10.1016/S0019-9958(83)80022-9]

[17] Esparza J, Ganty P, Kiefer S, Luttenberger M. Parikh's theorem: A simple and direct automaton construction. Information Processing Letters, 2011, 111(12): 614–619. [doi: 10.1016/j.ipl.2011.03.019]

[18] Ding RJ, Li GQ. Efficient implementation of coverability verification on communication-free Petri net. Ruan Jian Xue Bao/Journal of Software, 2019, 30(7): 1939−1952 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5750.htm [doi: 10.13328/j.cnki.jos.005750]

附中文参考文献:

[14] 杨启哲, 李国强. 基于通信Petri网的异步通信程序验证模型. 软件学报, 2017, 28(4): 804−818. http://www.jos.org.cn/1000-9825/5191.htm [doi: 10.13328/j.cnki.jos.005191]

[18] 丁如江, 李国强. 非交互式Petri网可覆盖性验证的高效实现. 软件学报, 2019, 30(7): 1939−1952. http://www.jos.org.cn/1000-9825/5750.htm [doi: 10.13328/j.cnki.jos.005750]

吴志文(1998－), 男, 硕士生, 主要研究领域为形式化验证, 程序分析与验证.

李国强(1979－), 男, 博士, 副教授, CCF 高级会员, 主要研究领域为形式化方法, 程序语言理论.

© 中国科学院软件研究所 http://www.jos.org.cn