软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*, 2022,33(8):2782−2796 [doi: 10.13328/j.cnki.jos.006598]

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

# 基于基本并行进程的异步通信程序的验证方法*

赵 櫻, 谭锦豪, 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**: 异步通信程序是进程间通过异步消息通信实现非阻塞并发的程序. 当前异步通信程序的程序验证问题通常将其归约至向量加法系统及其扩展模型, 因而复杂度很高, 缺乏高效工具. 基本并行进程作为向量加法系统的一个子类, 其可达性的验证问题为 NP 完备. 首先, 改进了 Osualdo 等人提出的为异步通信程序建模的 Actor 通信系统, 将其归约至基本并行进程. 然后, 实现了基本并行进程的模型检测工具 RABLE, 实验结果表明, 验证方法在异步通信程序的一系列程序验证问题上具有比已有工具更高效的结果.

**关键词**: 异步通信程序; 基本并行进程; Actor 通信系统; 模型检测; 可达性

**中图法分类号**: TP311

中文引用格式: 赵櫻, 谭锦豪, 李国强. 基于基本并行进程的异步通信程序的验证方法. 软件学报, 2022, 33(8): 2782−2796. http://www.jos.org.cn/1000-9825/6598.htm

英文引用格式: Zhao Y, Tan JH, Li GQ. Verification Method of Asynchronously Communicating Programs Based on Basic Parallel Processes. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2782−2796 (in Chinese). http://www.jos.org.cn/1000-9825/6598.htm

## Verification Method of Asynchronously Communicating Programs Based on Basic Parallel Processes

ZHAO Ying, TAN Jin-Hao, LI Guo-Qiang

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

**Abstract**: Asynchronously communicating program is the program that processes achieve non-blocking concurrency through asynchronous message passing. At present, the verification problem of asynchronously communicating program is usually reduced to vector addition system and its extension model, so it has high complexity and lack of efficient tools. Basic Parallel Processes, as a subclass of vector addition system, whose verification of complexity reachability is NP-complete, can also be used as an important model for verifying concurrent programs. Firstly, improve the Actor communicating system proposed by Osualdo, *et al*., by reducing it to Basic Parallel Processes. Then, realizing an automatic model checker for basic parallel processes named RABLE. The experimental results show that the verification method is more efficient than the existing tools for a series of program verification problems of asynchronously communicating programs.

**Key words**: asynchronously communicating program; basic parallel processes; actor communicating system; model checking; reachability

随着分布式计算机系统的发展, 异步通信相比同步实时通信显现出其实现简易的优势, 被越来越广泛应用. 非阻塞的异步通信并发是一种普遍的编程范式, 即调用方非阻塞地调用函数之后无需等待结果返回就能执行后续操作. 异步通信程序通过进程间进行异步通信以实现并发, 支持异步通信并发程序的语言主要有 Scala、Erlang[1], 以 Erlang 高阶函数式语言为例, 其程序不支持变量, 通过函数的递归取代循环, 闭包等信息通过函数参数或返回值进行传递, 过程中的状态变化高度依赖消息通信. 异步通信程序的编程思想源于 Actor 模型[2,3], 根据 Actor 模型, 异步通信程序的并发实质是一个通过消息传递进行通信的动态进程网络, 其中每

* 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: 2021-09-04; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

个进程可以执行 3 种基本操作: (1) 创建有穷数量的新进程; (2) 向某个进程发送有穷个消息; (3) 接收来自其他进程的消息. 每个进程有唯一的标识符, 并配置一个无界的邮箱缓存消息, 通过消息的模式匹配进行消息的搜索和消耗而无需等待结果回复, 于是异步通信程序的核心就是进程间依赖消息传递的通信手段, 实现进程的异步非阻塞并发. 以上性质和要求决定了异步通信程序的复杂度, 异步通信下推系统是一个模拟异步通信程序的程序模型, 其将每个有固定大小邮箱的可递归的进程都视作一个下推系统, 可得异步通信程序是图灵等价的[4], 因此很多如可达性的程序性质在其上不可判定[5].

目前针对异步通信程序的验证, 国内外研究主要通过近似或抽象的方法, 对异步通信程序进行限制从而建模并验证. 绝大多数的研究通过归约到 Petri 网或与其等价的向量加法系统. Sen 和 Viswanathan 提出了对异步通信下推系统进行空栈限制的多重集下推系统[6], 即只有下推进程的栈为空才能接收消息, Emmi 等人提出事件驱动的异步程序模型将数据 Petri 网的一个子类 v-PN 的可覆盖性归约到多重集下推系统的可达性, 得出多重集下推系统的状态可达性是可判定的[7]. 针对多重集下推系统, 有学者继续增加上下文限界限制, 限制程序在执行路径中的上下文切换次数, 这种限制是下近似[8]. 另一方面, 针对空栈限制过强, 不适用 Erlang 等依赖通信的程序的问题, Kochems 提出了更宽松的 K-shaped 限制, 限制了任意时间任意进程的栈上进行接收操作的栈符号个数, 栈同时包含递归和通信的信息, 同时提出了新的理论模型——Petri 网的扩展模型嵌套着色库所网(nets with nested colored tokens, NNCT), 证明了其等价于 *K*-shaped 限制下的异步通信下推系统[9]. Osualdo 等人将异步通信下推系统的进程限制为有穷状态进程, 提出了与向量加法系统等价的 Actor 通信系统, 并创立了一种针对 Erlang 程序的自动验证技术.

虽然向量加法系统的可达性可判定, 但其复杂度很高, 目前的结论是该问题存在一个 ACKERMANN 上界[10]和一个 Tower 难下界[11], 对于可覆盖性和有界性, 其复杂度为 EXPSPACE 完备[12,13]. 对于 NNCT 可覆盖性是 Tower 完备的, 有界性和终止性是 Tower 难[4]. 由于验证复杂度过高, 目前少数已经实现了的自动化求解工具如 BFC[14]、Petrinizer[15]也无法有效针对大规模的程序, 工具难以做到完备又高效, 因此多数研究仍停留在理论阶段.

针对以上问题, 本文给出了基于基本并行进程的异步通信程序的验证方法. 首先精化了与向量加法系统等价的 Actor 通信系统这一异步通信程序的程序模型, 通过在部分规则上修改语义使其满足基本并行进程[16]的定义要求, 将 Actor 通信系统归约到基本并行进程. 在 Actor 通信系统上形式化定义了“状态可达性问题”和“邮箱 *K* 限界问题”这两类安全性问题, 并将其转化为基本并行进程的可达性验证问题, 相比当前研究中的程序模型降低了异步通信程序的验证复杂度. 对于复杂度为 NP 完备的基本并行进程可达性问题, 根据 Verma 等人提出的归约算法可将该问题归约到存在性 Presburger 公式的可满足性[17], 我们借助 Z3 SMT 求解器的线性整数算术库实现该归约算法, 结合文献[18–20]已给出的基本并行进程的限界活性的归约算法, 开发完成了基本并行进程的可达性与限界活性验证工具 RABLE(reachability and bounded liveness for Erlang), 该工具能够验证基本并行进程的限界活性与可达性结果的同时, 还可返回求解的约束和模型的赋值信息. 最后对 Erlang 程序进行基于基本并行进程的 Actor 通信系统语义建模和安全性验证, 实验结果表明, 验证方法在异步通信程序的一系列程序验证问题上具有比已有工具更高效的结果, 在求解时间和生成约束的数量上都具有优势.

本文第 1 节概述预备知识, 包括基本并行进程和向量加法系统. 第 2 节给出异步通信程序的程序模型 Actor 通信系统和安全性性质相关的两类问题的定义. 第 3 节给出我们赋予的 Actor 通信系统在基本并行进程上的语义, 证明该语义是原始语义的上近似. 第 4 节对一个 Erlang 异步通信程序建模并验证. 第 5 节展示我们实现的可达性与限界活性验证工具 RABLE, 给出实验结果及分析. 第 6 节总结全文并探讨未来工作方向.

## 1 预备知识

### 1.1 基本并行进程

基本并行进程(basic parallel processes)是表示异步并行系统的一种基本模型, 它将一个并行系统的进程被建模为一个符号(symbol), 一个状态被建模为多个符号的并行组合, 一个符号可以通过迁移可以产生更多符

号, 通常基本并行进程产生一个无限状态系统. 一个基本并行进程的表达式包含了动作前缀、选择、结合运算, 分别表示进程的迁移、进程对迁移的选择和进程的并行组合. 基本并行进程中的迁移语义是异步的, 文献[21]中将基本并行进程视为交换上下文无关语法, 本文将采用这种定义.

令 $Var=\{X,Y,Z,...\}$ 和 $Act=\{a,b,c,...\}$ 分别为符号和动作的集合.

**定义 1(基本并行进程)**. 基本并行进程 BPP 是一个二元组 $(V,\Delta)$, 其中, $V\in Var$ 是一个符号的有穷集, $\Delta$是一个由规则组成的有穷集. $\Delta$中的规则的形式为 $X\xrightarrow{a}\alpha$,, 其中, $X\in V$, $a\in Act$, $\alpha\in V^{\oplus}$. $V^{\oplus}$ 表示由 $V$ 产生的自由交换幺半群(free commutative monoid). 一个 BPP $(V,\Delta)$ 规定了一个标签迁移系统 $(V^{\oplus},Act,\rightarrow,\alpha)$, 状态空间为 $V^{\oplus}$, 迁移关系为$\rightarrow$, 由以下规则产生:

$$\frac{X\xrightarrow{a}\alpha\in\Delta}{\beta X\gamma\xrightarrow{a}\beta\alpha\gamma}.$$

本文中为了简洁我们称 $V^{\oplus}$ 为 BPP 表达式, 其中的元素我们以$\alpha$、$\beta$、$\gamma$来表示.

记$\rightarrow^{*}$为单步迁移关系 $\{\xrightarrow{a}\}_{a\in Act}$ 的自反传递闭包, 则$\alpha\rightarrow^{*}\beta$表示状态$\alpha$经过若干($\geqslant 0$)次迁移可以到达$\beta$.

同时, BPP 的表达式具有模可交换性, 例如 XYZ、YXZ、ZXY 都被视为同一个元素, 因此直观上 BPP 表达式就是符号的并行组合, 每个符号能根据自己的规则独立地进行迁移.

可达性是形式化验证中模型的重要性质, 是本文研究的重点. 关于基本并行进程的可达性问题的复杂度, Esparza 证明了以下定理.

**定理 1(定理 3.2[22])**. 基本并行进程上的可达性问题是 NP 完备的.

### 1.2 向量加法系统

向量加法系统(vector addition system, VAS)是 Karp 和 Miller 提出的数学模型[23], 具有简洁的数学描述形式, 由于并发系统的状态和迁移可以用向量描述, 向量加法系统在并发程序的验证中有一定应用价值.

**定义 2(向量加法系统)**. 一个向量加法系统是一个序对$(I,R)$, 其中, $I$ 是一个有穷下标集(也称为 VAS 的格局), $R\in\mathbb{Z}^{I}$ 是一个有穷的迁移规则集.每条规则 $r\in R$ 是一个维度为$|I|$的整数向量.

一个向量加法系统$(I,R)$规定了一个状态空间为 $\mathbb{N}^{I}$ 的迁移系统 $\{(x,x+r)\mid x\in\mathbb{N}^{I},r\in R,x+r\in\mathbb{N}^{I}\}$. 例如, 若 $V=\{X_1,X_2,X_3\}$, BPP 表达式 $\alpha=X_1X_2X_3$, 则$\alpha$可以表示为向量(1,2,0).

基本并行进程是特殊的向量加法系统. 在给出了向量加法系统语义后, 我们可以将一个 BPP 表达式用更简洁的向量表示: 给定一个 BPP $(V,\Delta)$ 和 BPP 表达式$\alpha$, $V=\{X_1,...,X_n\}$, 我们建立 Parikh 映射 $P:V^{\oplus}\rightarrow\mathbb{N}^{n}$, 即令 $P(\alpha)=(\alpha(X_1),...,\alpha(X_n))$, 其中, $\alpha(X_1)$ 表示 $X_1$ 在$\alpha$中出现的次数. 由于基本并行进程限制了在每条规则中, 值为负数的分量最多只有一个, 且该分量的值只能为−1, 我们可以如下将该 BPP 转换为向量加法系统$(I,R)$: $I$=$V$, 且对于每条规则 $X\rightarrow\alpha\in\Delta$, 转化为向量加法系统的一个规则: 将 $P(\alpha)$中的$\alpha(X)$项减去 1.

可覆盖性是形式化验证中模型的重要性质, 也是本文研究的重点. 关于向量加法系统的可覆盖性问题的复杂度, Rackoff 证明了以下定理.

**定理 2(定理 3.5[12])**. 向量加法系统的可覆盖性问题复杂度是 EXPSAPCE 完备的.

## 2 Actor 通信系统

Actor 通信系统是由 Osualdo 等人提出, 模拟有限个进程通过无界邮箱并发通信的抽象模型, 被证明是 Erlang 程序的可靠模型[22], 刻画了并发异步通信中的 Actor 模型, 具有创建进程、发送消息和接收消息的行为.

### 2.1 Actor通信系统的定义

**定义 3(Actor 通信系统)**. Actor 通信系统(actor communicating system, ACS) $A$ 可以用一个四元组 $(Q,P,M,R)$ 来表示, 其中,

- $Q$ 是一个有穷的控制状态集;
- $P$ 是一个有穷的进程集;

• $M$ 是一个有穷的消息集;

• $R\subseteq Q\times Operations\times Q$ 是一个由有穷条转移规则构成的集合.

对于一条规则 $r\in R$, $r$ 是一个三元组 $(q_1,op,q_2)$, 通常记为 $q_1\xrightarrow{op}q_2$, $q_1,q_2\in Q$, $op\in Operations$.

给定 $p\in P$, $m\in M$, $Operations$ 包含 4 种操作: nop (dummy, 进程内部操作), $vq_0$ (spawn, 创建一个初始状态为 $q_0$ 的新进程, $q_0\in Q$), $p!m$ (send, 发送消息 $m$ 到进程 $p$), $p?m$ (receive, 从进程 $p$ 中接收消息 $m$).

ACS 模型具有无穷的状态, 因为一个进程的邮箱容量是无界的, 并且通信系统中的进程数量也可以很大, 但其进程的 id 集合是有穷的.

### 2.2 Actor通信系统的语义

在语义方面, Actor 通信系统不需要关注进程的邮箱中消息的顺序, 而是通过在邮箱上使用计数器抽象, 记录邮箱中消息数量. 并在每个进程类的控制状态上使用第 2 个计数器, 对处于当前控制状态的进程进行计数. 因此直观上, 我们使用支持计数的向量加法系统来表达 Actor 通信系统的语义.

**定义 4(Actor 通信系统的语义)**. Actor 通信系统 $A=(Q,P,M,R)$ 的语义是由向量加法系统 $\gamma=(I,R)$. 生成的转移系统, 格局集 $I=Q\cup(P\times M)$, 一个格局 $c$ 可以用向量组 $(u,v)\in\mathbb{N}^I$ 表示, 其中, $|u|=|Q|$ 且 $|v|=|P\times M|$, 其中向量 $u=(q_1,q_2,...,q_{|Q|})$ 包含对所有状态的计数, $v=((p_1,m_1),(p_1,m_2),...)$ 包含所有进程中各消息的计数, 引入记号 $u[q_1]$ 表示状态 $q_1$ 出现的次数, 引入记号 $v[(p_1,m_1)]$ 表示进程 $p_1$ 的邮箱中 $m_1$ 的计数. 定义格局之间的转移关系 $c_1\overset{r}{\Rightarrow}c_2$, 对于 Actor 通信系统的每条规则 $r=q_1\xrightarrow{op}q_2\in R$, 根据操作$op$的类型, 我们分别定义向量加法系统的规则 $\boldsymbol{R}$.

- 若 $op$=$nop$, 则 $u[q_1]-1$, $u[q_2]+1$;
- 若 $op$=$vq_3$, 则 $u[q_1]-1$, $u[q_2]+1$, $u[q_3]+1$;
- 若 $op$=$p!m$, 则 $u[q_1]-1$, $u[q_2]+1$, $v[(p,m)]+1$;
- 若 $op$=$p?m$, 且 $v[(p,m)]>0$, 则 $u[q_1]-1$, $u[q_2]+1$, $v[(p,m)]-1$.

定义$\Rightarrow$为 $\bigcup\limits_{r\in R}\overset{r}{\Rightarrow}$, 并记$\overset{*}{\Rightarrow}$为$\Rightarrow$的自反传递闭包.

为了展示 Actor 通信系统的表达能力, 我们举一个异步通信的 Erlang 程序实例进行说明. 如下是一个 Erlang 程序实例, 程序入口是 *start* 函数, 有两个循环的函数体 *loop_a* 和 *loop_b*, 起始进程是 *A*, 进程 *A* 在 *start* 函数中会创建一个新的进程 *B* 并让其执行 *loop_b*(*A*), 然后进程 *A* 执行 *loop_a*(*B*). 执行 *loop_a*(*B*)的函数体时, 进程 *A* 发送一条消息 *msg*1 给进程 *B*, 然后等待 *msg*2, 当接收到 *msg*2 后从头执行该循环. 进程 *B* 执行函数体 *loop_b*(*A*)时需要等待 *msg*1, 在接收 *msg*1 后会发送消息 *msg*2 给进程 *A*, 然后重新执行该循环.

```
1  start() ->
2      B = entry(self()),
3      loop_a(B).
4
5  loop_a(P) ->
6      P ! msg1,
7      receive
8          msg2 -> loop_a(P)
9      end.
10
11 entry(A) ->
12     spawn(fun()-> loop_b(A) end).
13
14 loop_b(P) ->
15     receive
16         msg1 ->
17             P ! msg2,
18             loop_b(P)
19     end.
```

接下来, 我们用 Actor 通信系统 *A* 来模拟以上 Erlang 程序, 图 1 用有向图展示了对应的 Actor 通信系统 *A*

的规则集; $A$ 的控制状态集为 $\{q_A, q'_A, q''_A, q_B, q'_B\}$, 消息集 $M=\{m_1, m_2\}$, 进程集 $P=\{p_A, p_B\}$, 图中点和边分别为控制状态和规则. 例如, 从 $q'_A$ 到 $q''_A$ 标记为 $p_B!m_1$ 的边即表示规则 $q'_A \xrightarrow{p_B!m_1} q''_A$, 对应 Erlang 程序第 6 行进程 $A$ 发送消息 *msg*1 给进程 *B*. 此处为了简洁性, 在图 1 中, 我们省略了与 *nop* 操作相关的边.

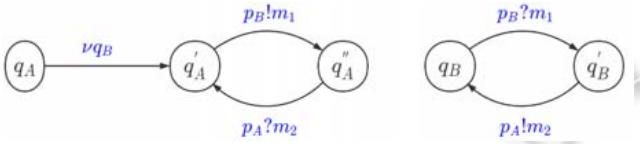


图 1 Actor 通信系统的迁移规则

图 2 为由 Actor 通信系统$\mathcal{A}$生成的转移系统示意图. 图中的每个点代表一个格局, 边表示格局转移的规则, 我们用符号||分隔了状态计数器和消息计数器. 具体计数情况如下; $q_A$ 表示起始格局 $(u_0, v_0)$, 满足 $u_0[q_A]=1$, 对于其他任意状态 $q$, $u_0[q]=0$. 对任意 $p\in P$, $m\in M$, $v_0[(p,m)]=0$. 图中被标记为 $q''_A \| q_B \| (p_B, m_1)$ 的点设其为格局 $(u_2, v_2)$, 其满足 $u_2[q''_A]=u_2[q_B]=1$, $v_2[(p_B, m_1)]=1$, 对于其他任意 $q\in Q$, $p\in P$, $m\in M$, $u_2[q]=0$ 且 $v_2[(p,m)]=0$.

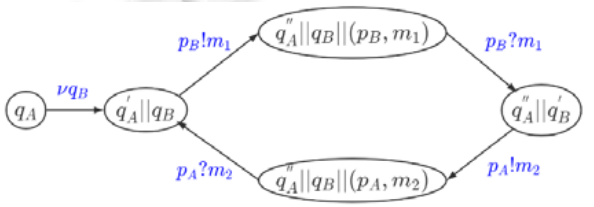


图 2 由 Actor 通信系统生成的转移系统

### 2.3 验证问题及形式化定义

Actor 通信系统是 Erlang 程序的可靠模型, 它的语义包含了 Erlang 程序的行为. Actor 通信系统的语义中包含两个计数器, 能够分别对程序控制状态和信道中的消息数量进行计数, 通过这两个计数器, 我们能借助 Actor 通信系统对异步通信程序的安全性性质进行验证, 本文针对此将重点讨论两个程序安全性问题; 程序中错误状态的可达性问题以及邮箱的限界问题.

#### 2.3.1 Actor 通信系统的状态可达性问题

在实际编写程序时, 由于语法错误和逻辑错误等原因, 程序中有可能会出现预期之外的错误或异常行为. 由于异步通信程序的行为和并发模式复杂, 因此设计软件测试用例变得困难. 通过形式化验证中的模型检测技术, 将程序转化为抽象模型, 能够检测该模型是否可达一些错误或异常状态, 进而自动帮助我们找到导致程序进入错误状态的因素, 也称之为反例(counter example).

我们可以通过验证 Actor 通信系统的可覆盖性来证明错误状态不可达. 于是引入 Actor 通信系统的状态可达性(reachability)验证问题.

**定义 5(Actor 通信系统的状态可达性问题)**. 给定一个 Actor 通信系统 $A=(Q,P,M,R)$, 一个格局 $c_0$, 以及 $k\in\mathbb{N}$, $q\in Q$, 状态可达性问题为: 是否存在一个格局 $c=(u,v)$, 使得 $u[q]\geqslant k$ 且 $c_0 \Rightarrow^* c$.

#### 2.3.2 Actor 通信系统的邮箱 *K* 限界问题

对于异步通信程序而言, 邮箱(mailbox) 是存储消息的重要数据结构, 因此本文关注了与邮箱密切关联的内存性能问题. Erlang 程序中的邮箱是进程用来存储尚未处理的消息的容器, 一般不限容量, 换言之, 一个进程可以不断接收消息直至内存耗尽. 这种情况下为了加快消息的消耗速度, 提出了一些技术方法, 如将消费者的工作量转移到生产者、消息批处理、限制消息接收速率、将消息写入内存之外的空间减少负载等. 因此避免内存浪费和提升性能, 邮箱的内存利用也是异步通信中值得关注的问题.

以图 2 的 Actor 通信系统为例, 可知对于进程 $A$ 每接收一条消息就会进入下一状态, 进程 $A$ 的邮箱中任

意时间至多有 1 条消息, 也即不存在一个格局 $c=(u,v)$, 使得 $v[(p_A,m_1)]\geqslant 2$, 由此我们就只需要分配 1 块消息内存给进程 $A$.

因此为了防止邮箱占用过多的内存, 我们可以预先给定程序中一个进程邮箱可容纳消息个数的上界. 然后对一个进程 $p$, 判断是否可能到达 $p$ 中消息数量多于 $K$ 的状态, 如果不可达, 那么只需要分配$K$块内存给进程 $p$. 这样我们将邮箱内存问题归约到了可达性问题上, 通过验证 Actor 通信系统的可达性来证明实际 Erlang 程序中邮箱存储的消息个数存在一个上界. 于是引入 Actor 通信系统的邮箱 $K$ 限界验证问题.

**定义 6(Actor 通信系统的邮箱 $K$ 限界问题)**. 给定一个 Actor 通信系统 $A=(Q,P,M,R)$, 一个格局 $c_0$, 以及 $k\in\mathbb{N}$, $p\in P$, 邮箱 $K$ 限界问题为: 是否存在一个格局 $c=(u,v)$ 使得 $|v(p)|=\sum_{m\in M}v[(p,m)]\geqslant K$, 且 $c_0\Rightarrow^* c$.

以上两个安全性问题可以通过 Actor 通信系统的可覆盖性来验证, 根据定理 2, Rackoff 已经证明了与其等价的向量加法系统可覆盖性问题复杂度是 EXPSAPCE 完备的[9].

## 3 基本并行进程的语义

由于 Actor 通信系统的可达性的验证复杂度是 EXPSPACE 完备的, 为了降低验证复杂度, 我们提出了基于基本并行进程的 Actor 通信系统的验证方法. 先赋予 Actor 通信系统基于基本并行进程的新语义, 引入新标签 *in* 和 *out* 分别记录消息进入和离开信道的行为. 证明了 Actor 通信系统的基本并行进程语义是对原始语义的一个上近似, 即包含了比原始语义更多的行为, 这样当基本并行进程中的可达性不满足时, Actor 通信系统中对应的状态也不可达, 于是可以利用基本并行进程的语义更高效地验证 Actor 通信系统的两类问题, 给出了两类安全性问题的 BPP 形式.

### 3.1 Actor通信系统的新语义

在 Actor 通信系统的语义中, 针对规则 $q_1\overset{p?m}{\rightarrow}q_2\in R$, 有两个符号减少的情况.

(1) 因为 $q_1$ 转化为 $q_2$, 所以当前格局中状态 $q_1$ 的数量要减 1;

(2) 副作用 $p?m$ 会从邮箱中消耗消息 $m$, 因此邮箱 $p$ 中的消息 $m$ 数量要减 1.

此时有两个不同符号的数值要同时减少, 而基本并行进程的规则左侧只允许出现 1 个符号, 因此若用基本并行进程来模拟 Actor 通信系统, 必须避免同时消耗两个不同的符号. 下面引入基于 BPP 的 Actor 通信系统新语义, 修改方式是对每条消息增加两个标签 *in* 和 *out*, 分别记录该消息进入和离开邮箱的历史. 将格局集中邮箱和消息的映射从原本的 $P\times M$ 改为 $P\times(M\times\{in,out\})$, 修改原语义中两条规则 $q_1\overset{p!m}{\rightarrow}q_2$ 和 $q_1\overset{p?m}{\rightarrow}q_2$ 如下.

- 若 $op=p!m$, 则 $u[q_1]-1$, $u[q_2]+1$, $v[(p,m^{in})]+1$;
- 若 $op=p?m$, 则 $u[q_1]-1$, $u[q_2]+1$, $v[(p,m^{out})]+1$.

给出了从原语义格局转换成新语义格局的算法, 称为格局转换函数, 形式化定义如下.

**定义 7(格局转换函数)**. 给定一个 Actor 通信系统 $A=(Q,P,M,R)$, 其中, $|Q|=x\in\mathbb{N}$, 输入原语义下的格局 $(u_A,v_A)$ 到格局转换函数, 将输出一个新语义下的格局 $(u_B,v_B)$ 满足:

- 对 $\forall 1\leqslant i\leqslant x$, $u_B[q_i]=u_A[q_i]$;
- 对 $\forall p\in P$, $m\in M$, $v_B[(p,m^{in})]=v_A[(p,m)]$;
- 对 $\forall p\in P$, $m\in M$, $v_B[(p,m^{out})]=0$.

关于 Actor 通信系统新语义和原语义的区别: 新语义下状态计数器的含义不变, 消息计数器的含义和计数方法都发生改变. 每个消息带有 *in* 或 *out* 的标签后, 消息计数器对消息的进入和离开进行计数. 规则上的区别在于: 对于进程 $p$ 和消息 $m$, $p$ 的邮箱中 $m^{in}$ 的数量为原语义里 $p$ 中 $m$ 的数量, $p$ 中 $m^{out}$ 起始计数为 0, 而当进程 $p$ 消耗消息 $m$ 时, 原语义减少了 $p$ 中 $m$ 的个数, 新语义则是增加 $p$ 中 $m^{out}$ 的计数.

关于Actor通信系统新语义和原语义的关系: 可以证明我们给出的Actor通信系统新语义是原语义的上近

似, 即包含了更多行为. 下面给出的定理 3 证明了, 对于任意在原语义下可达的格局 $c_A$, 存在新语义下的一个可达格局 $c_B$, $c_B$ 中各个状态的数量与 $c_A$ 一致, 而 $c_B$ 中 $(p,m^{in})$ 与 $(p,m^{out})$ 的数量之差即 $c_A$ 中$(p,m)$的出现次数.

**定理 3**. 一个 Actor 通信系统 $A=(Q,P,M,R)$, 其中, $|Q|=x$, $|P|=y$, $|M|=z$, 且 $x, y, z\in\mathbb{N}$, 给定一个原语义下的格局 $c_{A_0}$, 若存在 $c_A=(u_A,v_A)$ 满足:

- 对 $\forall 1\leqslant i\leqslant x$, $u_A[q_i]=k_i$;
- 对 $\forall 1\leqslant i\leqslant y$, $v_A[p_i]=m_1^{h_{i1}}...m_z^{h_{iz}}$, $v_A[p_i]$ 表示 $p_i$ 的消息序列;
- $c_{A_0}\Rightarrow^* c_A$,

则存在新语义下的一个格局 $c_B$ 使得:

- 对 $\forall 1\leqslant i\leqslant x$, $u_B[q_i]=k_i$;
- 对 $\forall 1\leqslant i\leqslant y$, $v_B(p_i)=m_1^{in^{r_{i1}}}m_1^{out^{s_{i1}}}...m_z^{in^{r_{iz}}}m_z^{out^{s_{iz}}}$, $v_B(p_i)$ 表示 $p_i$ 的消息序列;
- 对 $\forall 1\leqslant i\leqslant y$, $\forall 1\leqslant j\leqslant z$, $r_{ij}-s_{ij}=h_{ij}$;
- $c_{B_0}\Rightarrow^* c_B$, $c_{B_0}$ 为 $c_{A_0}$ 输入格局转换函数后输出的格局.

证明:

使用归纳法对定理 3 进行正确性证明. 基础步: $c_A$ 即 $c_{A_0}$, 输入格局转换函数得到的 $c_{B_0}$ 就是新语义下的格局. 归纳步: 为不失一般性, 令 $r=q_1\xrightarrow{op}q_2$, 且 $c_{A_0}\Rightarrow^* c'_A\overset{r}{\Rightarrow}c_A$, 接下来对$op$的操作类型进行分类讨论.

- 若 $op$=$nop$, 根据规则 $c'_A=(u'_A,v'_A)$ 满足 $u'_A[q_1]=k_1+1$, $u'_A[q_2]=k_2-1$, 根据归纳假设, 存在新语义下的格局 $(u'_B,v'_B)$ 使得 $(u_{B_0},v_{B_0})\Rightarrow^*(u'_B,v'_B)$, 且根据语义 $u'_B=u'_A$, 因此 $u'_B[q_1]=u'_A[q_1]=k_1+1\geqslant 1$, 可得 $(u'_B,v'_B)\overset{r}{\Rightarrow}(u_B,v_B)$, 其中, $u_B$ 需要满足 $u_B[q_1]=u'_B[q_1]-1=k_1$, 且 $u_B[q_2]=u'_B[q_2]+1=k_2$. 除此之外, 由于 $nop$ 无其余副作用, 格局中其余元素相等, 因此格局 $c_B=(u_B,v_B)$ 符合条件.
- 若 $op=vq_3$, 根据规则 $c'_A=(u'_A,v'_A)$ 满足 $u'_A[q_1]=k_1+1$, $u'_A[q_2]=k_2-1$, $u'_A[q_3]=k_3-1$, 根据归纳假设, 存在新语义下的格局 $(u'_B,v'_B)$ 使得 $(u_{B_0},v_{B_0})\Rightarrow^*(u'_B,v'_B)$, 且根据语义 $u'_B=u'_A$, 可得 $(u'_B,v'_B)\overset{r}{\Rightarrow}(u_B,v_B)$, 其中, $(u_B,v_B)$ 需满足: $u_B[q_1]=u'_B[q_1]-1=k_1$, $u_B[q_2]=u'_B[q_2]+1=k_2$, $u_B[q_3]=u'_B[q_3]+1=k_3$. 除此之外, $(u_B,v_B)$ 与 $(u'_B,v'_B)$ 一致, 因此格局 $c_B=(u_B,v_B)$ 符合条件.
- 若 $op=p_i!m_j$, 根据规则 $c'_A=(u'_A,v'_A)$ 满足 $u'_A[q_1]=k_1+1$, $u'_A[q_2]=k_2-1$, $v'_A[(p_i,m_j)]=h_{ij}-1$, 除此之外, $(u_A,v_A)$ 与 $(u'_A,v'_A)$ 一致. 根据归纳假设, 存在新语义下的格局 $(u'_B,v'_B)$ 使得 $(u_{B_0},v_{B_0})\Rightarrow^*(u'_B,v'_B)$, 且根据语义 $u'_B=u'_A,$, $v'_B[(p_i,m_j^{in})]-v'_B[(p_i,m_j^{out})]=h_{ij}-1$.

因为 $u'_B[q_1]=u'_A[q_1]=k_1+1\geqslant 1$, 可得 $(u'_B,v'_B)\overset{r}{\Rightarrow}(u_B,v_B)$, 其中, $(u_B,v_B)$ 需满足 $u_B[q_1]=u'_B[q_1]-1=k_1$ 且 $u_B[q_2]=u'_B[q_2]+1=k_2$, 且 $v_B[(p_i,m_j^{in})]=v'_B[(p_i,m_j^{in})]+1$. 除此之外格局元素一致, 可得:

$$v_B[(p_i,m_j^{in})]-v_B[(p_i,m_j^{out})]=v'_B[(p_i,m_j^{in})]+1-v'_B[(p_i,m_j^{out})]=h_{ij}-1+1=h_{ij}=v_A[(p_i,m_j)],$$

$(u_{B_0},v_{B_0})\Rightarrow^*(u_B,v_B)$ 得证, 因此格局 $c_B=(u_B,v_B)$ 符合条件.

- 若 $op=p_i?m_j$, 根据规则 $(u'_A,v'_A)$ 满足 $u'_A[q_1]=k_1+1$, $u'_A[q_2]=k_2-1$, $v'_A[(p_i,m_j)]=h_{ij}+1$, 除此之外, $(u_A,v_A)$ 与 $(u'_A,v'_A)$ 一致. 根据归纳假设, 存在新语义下的格局 $(u'_B,v'_B)$ 使得 $(u_{B_0},v_{B_0})\Rightarrow^*(u'_B,v'_B)$, 且根据语义 $u'_B=u'_A$, $v'_B[(p_i,m_j^{in})]-v'_B[(p_i,m_j^{out})]=h_{ij}+1$. 与 $op=p_i!m_j$ 时类似, 可得 $(u'_B,v'_B)\overset{r}{\Rightarrow}(u_B,v_B)$, 其中, $(u_B,v_B)$ 需满足 $u_B[q_1]=k_1$, $u_B[q_2]=k_2$, 且 $v_B[(p_i,m_j^{out})]=v'_B[(p_i,m_j^{out})]+1$. 除此之外格局元素一致, 可得:

$$\begin{aligned}
&v_B\left[(p_i,m_j^{in})\right]-v_B\left[(p_i,m_j^{out})\right]\\
&=v_B\left[(p_i,m_j^{in})\right]-(v'_B\left[(p_i,m_j^{out})\right]+1)\\
&=v_B\left[(p_i,m_j^{in})\right]-v'_B\left[(p_i,m_j^{out})\right]-1\\
&=h_{ij}+1-1\\
&=v_A\left[(p_i,m_j)\right].
\end{aligned}$$

$(u_{B_0},v_{B_0})\Rightarrow^*(u_B,v_B)$ 得证, 因此格局 $c_B=(u_B,v_B)$ 符合条件.

综上, 定理 3 得证. □

由定理 3 可知 Actor 通信系统新语义是原语义的上近似, 意味着其能包含比原语义更多的行为, 以下举例说明. 考虑一个 Actor 通信系统 $A=(Q,P,M,R)$, $Q=\{q\}$, $P=\{p\}$, $M=\{m\}$, $R$ 由两条规则组成: $r_1:q\xrightarrow{p!m}q$, $r_2:q\xrightarrow{p?m}q$, 假设起始格局 $c_{A_0}=(u_{A_0},v_{A_0})$, 其中, $u_{A_0}(p)=1$, $v_{A_0}\left[(p,m)\right]=0$, 即进程 $p$ 的初始邮箱为空. 直观来看, 原语义规则中的 $v\left[(p,m)\right]>0$ 要求了只有当邮箱中存在消息 $m$ 才能将其消耗, 因此 $c_{A_0}$ 只能够先使用规则 $r_1$ 使消息 $m$ 进入 $p$, 再使用规则 $r_2$ 将消息 $m$ 消耗. 而新语义不存在这样的限制, 允许了邮箱在未包含消息的情况下消耗消息, 即通过格局转换函数得到的格局 $c_{B_0}$ 能够使用 $r_1$ 和 $r_2$ 进行迁移. 尽管进程非法消耗消息的行为在新语义下是可达的, 多数情况下邮箱 $K$ 限界问题的条件也足够限制并验证程序.

### 3.2 Actor通信系统的基本并行进程语义

我们给出的 Actor 通信系统的新语义满足了基本并行进程的定义, 并且是原始语义的上近似关系, 因此可以使用基本并行进程表达等价的语义. 我们最终给出 Actor 通信系统的基本并行进程语义.

**定义 8(Actor 通信系统的基本并行进程语义)**. 给定一个 Actor 通信系统 $A=(Q,P,M,R)$, 其中, $|Q|=x$, $|P|=y$, $|M|=z$, 且 $x,y,z\in\mathbb{N}$. 我们构造 BPP $(V,\Delta)$, 其中, $V=Q\cup(P\times M\times\{in,out\})$. 对于每条规则 $q_1\xrightarrow{op}q_2\in R$ 根据操作 $op$ 的类型, 定义规则集 $\Delta$:

- 若 $op$=$nop$, 引入规则 $q_1\to q_2$;
- 若 $op$=$vq_3$, 引入规则 $q_1\to q_2\,\|\,q_3$;
- 若 $op$=$p!m$, 引入规则 $q_1\to q_2\,\|\,(p,m^{in})$;
- 若 $op$=$p?m$, 引入规则 $q_1\to q_2\,\|\,(p,m^{out})$.

可以将 Actor 通信系统的一个格局 $(u_B,v_B)$ 转化为 BPP 的一个表达式, 假设对 $\forall 1\leqslant i\leqslant x$, $u_B\left[q_i\right]=k_i$, 对 $\forall 1\leqslant i\leqslant y$, $\forall 1\leqslant j\leqslant z$, $v_B\left[(p_i,m_j^{in})\right]=r_{ij}$, $v_B\left[(p_i,m_j^{out})\right]=s_{ij}$, 则 $(u_B,v_B)$ 转移为 BPP 的表达式: $q_1^{k_1}\|...q_x^{k_x}\|(p_1,m_1^{in})^{r_{11}}\|...\|(p_y,m_z^{out})^{s_{yz}}$.

现给定一个 Actor 通信系统 $A=(Q,P,M,R)$ 和与之等价的 BPP $(V,\Delta)$ 表达式, 令 $\alpha_0$ 为 $A$ 的起始格局转换得到的 BPP 表达式, 令 $\alpha_s\in V^{\oplus}$, 根据定理 3, 有以下推论.

**推论 1**. 给定 $k\in\mathbb{N}$, $q\in Q$, 如果 $\alpha_s$ 满足 $\alpha_s[q]\geqslant k$, 并且从 $\alpha_0$ 出发, $\alpha_s$ 不可达, 则 Actor 通信系统的状态可达性问题的回答为否.

**推论 2**. 给定 $K\in\mathbb{N}$, $p\in P$, 如果 $\alpha_s$ 满足 $\sum_{m\in M}(\alpha_s[(p,m^{in})]-\alpha_s[(p,m^{out})])\geqslant K$, 并且从 $\alpha_0$ 出发, $\alpha_s$ 不可达, 则 Actor 通信系统的邮箱 $K$ 限界问题的回答为否.

## 4 Erlang 程序实例的验证

本节将以一个具体实例直观说明如何通过基本并行进程的可达性验证异步通信 Erlang 程序的安全性.

如下是一个 Erlang 程序 state_factory, 该 state_factory 程序启发于 Osualdo 等人对 Accumulator Factory 的泛化版本[1]. Accumulator Factory 程序问题要求创建一个函数, 该函数必须同样返回一个函数体, 实现每次调

用都对该参数进行一次累加或修改并返回. 由于 Erlang 不允许可变变量, 但是在异步请求的闭包中有变量捕获特性, 因此通过生成一个循环的进程, 获取变量后增加并将其返回给调用者, 从而模仿这种可变状态.

```
1  main() ->
2      StatefulFun = factory(?rand_nat(), fun(_X,_Y) -> ?rand_nat() end),
3      loop(?rand_nat(), StatefulFun).
4
5  loop(1,Fun) -> Fun(?rand_nat());
6  loop(N,Fun) -> Fun(?rand_nat()), loop(N-1,Fun).
7
8  state(N, NewState) ->
9      % Mailbox here won't have more than one message.
10     receive {A,In}->
11         % Mailbox here won't have any messages.
12             M = NewState(N,In),
13             A ! M,
14             state(M, NewState)
15     end.
16
17 factory(N, NewState) ->
18     B=spawn(fun() -> state(N,NewState) end),
19     fun(In)->
20             B!{self(),In},
21             receive
22                 Out-> Out
23             end
24     end.
```

该 state_factory 程序中, ?rand_nat()是一个宏, 作用是随机生成一个自然数, 此处省略具体实现, main 实例化 factory 和 loop. factory 创建执行有状态函数 state 的新进程, 将状态作为参数与其绑定, 并发送消息给进程. 接收消息后创建新的状态, 然后通过发送消息继续递归 state. loop 循环 factory 的行为从而实现进程的并发.

由于 state 等待消息, 并且只有在接收消息后创建新状态, 因此下面程序的第 9 行注释标出当前位置相应进程邮箱中消息数不会大于 1, 第 11 行注释标出当前位置消耗消息后相应进程邮箱中不会包含消息, 即这两处需要分别验证邮箱的限界问题, 若回答为否, 表示有多个消息请求来到第 9 或 11 行, 即同时进行状态的更新, 也就违反了每次调用只更新一次的互斥性的要求.

图 3 是 state_factory 程序的一个循环(即 loop 第 1 个参数为 1 时)生成的基于 BPP 的 Actor 通信系统, 我们以 $q_A$ 表示入口进程 $A$ 的初始状态, 以 $q_B$ 表示 $A$ 创建的进程 $B$ 的初始状态. 对应第 9 行和第 11 行注释给出的邮箱限界问题, 根据图中的 BPP 迁移规则, 可知不存在满足:

(1) $\alpha_s[q_B]\geqslant 1$, 且 $\sum_{m\in M}(\alpha_s[(p_B,m_1^{in})]-\alpha_s[(p_B,m_1^{out})])>1$;

(2) $\alpha_s[q_B']\geqslant 1$, 且 $\sum_{m\in M}(\alpha_s[(p_B,m_1^{in})]-\alpha_s[(p_B,m_1^{out})])>0$.

的可达 BPP 状态, 因此, 由 BPP 的可达性可以推断该 Actor 通信系统的上述两个验证问题的回答为否.

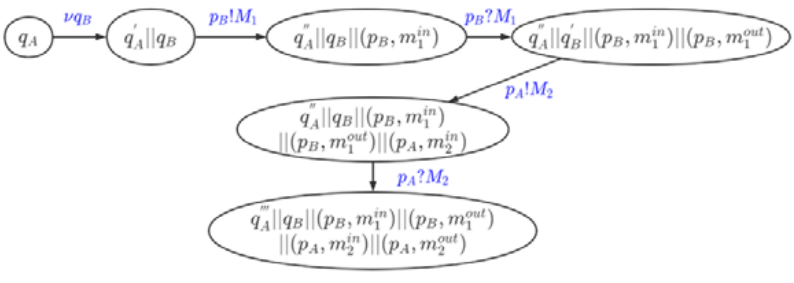


图 3 state_factory 生成的基于基本并行进程的 Actor 通信系统

在通过基本并行进程语义验证程序邮箱限界性质的同时, 还验证了任意时间点没有冲突请求在同时更新状态, 类似性质能够关联实际应用中的并发事务隔离性(isolation), 因此具有一定应用价值. 综上, state_factory 例子展示了 Actor 通信系统的基本并行进程语义对 Erlang 异步通信程序多种性质的表达能力, 即

高阶函数特性、消息传递、进程的动态创建、互斥(mutual exclusion)这类在并发程序中的重要安全性性质.

考虑到实际程序中包含更复杂的数据处理, 可以使用数据抽象技术给出内部计算概况, 在进程并行、消息通信时结合我们的模型. 例如在 state_factory 中, 若在进程 $B$ 的消息处理阶段增加不同 state 范围判断条件下的计算处理, 则图 3 的 Actor 通信系统将需要更多状态, 通过数据抽象解释可以将当前为随机自然数的 state 抽象为奇偶分析, 如图 4 所示, 从而将 state 变量限制到有限域, 限制新的状态数, 并进行后续验证分析.

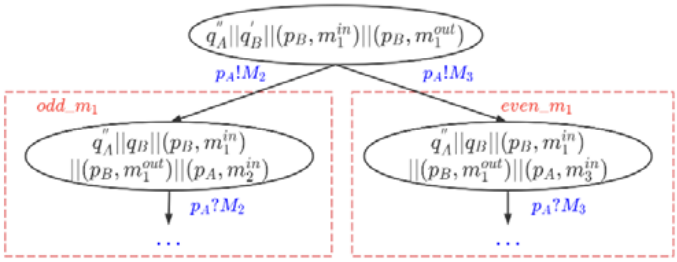


图 4 结合数据抽象解释后的 state_factory 部分状态示例

## 5 模型检测工具与实验结果

本节给出我们实现的基本并行进程模型检测工具 RABLE 的技术架构, 其具有检测可达性和限界活性的功能. 通过验证基本并行进程的可达性对 Actor 通信系统的安全性性质进行了验证. 实验结果表明, 我们的方法和工具比已有方法在求解时间和生成约束的数量上都更有优势.

### 5.1 基本并行进程的模型检测工具RABLE

我们实现了基本并行进程的限界活性与可达性模型检测工具 RABLE. 在限界活性方面, 由文献[18–20]贡献并完成了最初的算法, 通过将基本并行进程的 EG 逻辑限界模型检测归约到了线性整数算术公式的可满足性. 在此基础上, 我们又实现了 Verma 等人提出的算法[17], 将基本并行进程的可达性归约到存在性 Presburger 公式, 本节将着重介绍可达性检测部分的算法.

在求解引擎方面, 使用了 Z3 SMT 求解器. SMT 求解器是自动化求解 SAT 问题及其扩展 SMT 问题[24]的工具, 由微软公司主导开发的 SMT 求解器 Z3 所支持理论最多, 性能好[25]. RABLE 使用了 Python3 实现, 借助 SMT 求解器 Z3 的 Python 接口作为求解可满足性的引擎, 主要使用了其线性整数算术的库[20].

RABLE 架构如图 5 所示, RABLE 内包含 3 个模块: 语法分析器、基本并行进程模型以及 SMT 求解器.

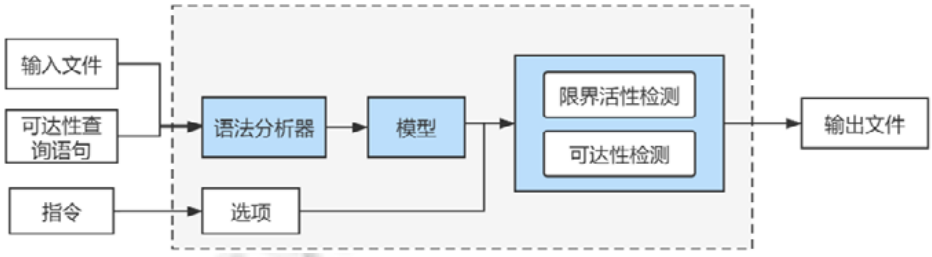


图 5 RABLE 的架构

#### 5.1.1 输 入

RABLE 的可达性查询输入为: 基本并行进程的变量、规则表达式, 和一个状态可达性的查询语句.

语法分析器的任务是检查用户输入是否符合基本并行进程的定义, 以及可达性查询语句的语法是否正确.查询语句的语法为 query $a_1x_1+\ldots+a_nx_n \lhd b$, 其中, $a_1,\ldots,a_n, b\in\mathbb{Z}$, $\lhd\in\{==,!=,>=,<=,>,<\}$. 如果输入不合法工具将返回错误提示并中止执行.

如果输入合法则根据输入构造基本并行进程模型, 生成相应的逻辑公式和约束, 输入到 SMT 求解器进行

可达性检测, 接下来我们重点介绍如何构造可达性的约束公式.

5.1.2 可达性约束构造

根据定理 1, Esparsa 已证明了基本并行进程的可达性验证是 NP 完备的, 目前的研究路线是通过将 NP 完备可达性检测问题归约更小的计算对象, 再通过高效的 SMT 求解器进行求解. Verma 等人证明了上下文无关语法的 Parikh 映像可以归约到存在性 Presburger 公式的可满足性[17]. 而这种归约存在一个问题, 使得构造出的公式约束过少, 对此 Barner 进行修正, 我们实现了由 Hague 等人提出的与 Barner 的版本逻辑等价的归约[26,27].

给定一个 BPP 表达式 $(V,\Delta)$ 和一个 BPP 初始表达式 $\alpha$. 我们假设 $\alpha$ 只含有一个符号, 假设 $\alpha = P_{init} \in V$. 下面介绍如何构造存在性 Presburger 公式, 首先引入变量:

- 对于任意进程符号 $P \in V$, 引入符号 $x_p$ 表示进程 $P$ 出现的次数.
- 对于任意迁移规则 $r \in \Delta$, 引入符号 $y_r$ 表示规则 $r$ 使用的次数.
- 根据迁移规则定义结点为进程符号的生成树, 对于任意进程符号 $P \in V$, 引入符号 $z_p$, 代表生成树中 $P$ 到 $P_{init}$ 的距离.

引入上述变量后, 我们可以构造存在性 Presburger 公式. 该公式主要有两部分: 第 1 部分主要表达了迁移规则的使用次数必须与 BPP 表达式中的符号数量一致. 例如, 若 $r: P \to YZ$ 是唯一产生符号 $Y$ 的规则, 则使用规则 $r$ 的次数必定等于 $Y$ 的出现次数, 即 $y_r$ 等于 $X_Y$. 该部分含有以下 3 类约束.

(1) 对于任意 $P \in V$, 引入约束: $x_p \geqslant 0$;

(2) 对于任意 $r \in \Delta$, 引入约束: $y_r \geqslant 0$;

(3) 对于任意 $P \in V$, 令 $r_1,...,r_k$ 为所有左边的进程符号为 $P$ 的规则, 引入约束:

$$\alpha(P) + \sum_{r \in \Delta} y_r r^{\cdot}(P) - \sum_{i=1}^{k} y_{r_i} = x_P.$$

需要注意的是第 1 部分的 3 类约束不足以表达可达性, 规则左右边的符号若相同将出现该规则的约束抵消的情况, 使得这样的规则可为任意值, 所以需要增加对各状态和产生其的规则被使用前提的描述, 即某些状态的可达前提是产生的规则被使用次数必须大于 0. 因此存在性 Presburger 公式的第 2 部分通过使用变量构造以下两类约束.

(1) 对于任意 $P \in V$, 引入约束: $x_P = 0 \vee z_P > 0$;

(2) 对于任意 $P \in V$, 令 $r_1,...,r_l$ 为所有右边出现进程符号 $P$ 的规则, 令 $Y_1,...,Y_l$ 为相应规则左边的进程符号, 引入约束:

$$\left(z_p = 0 \wedge \bigwedge_{i=1}^{l} y_{r_i} = 0\right) \vee \bigvee_{i=1}^{l} (z_p = z_{Y_i} + 1 \wedge y_{r_i} > 0 \wedge z_{Y_i} > 0).$$

若 $Y_i$ 为初始 BPP 状态 $P_{init}$, 则该公式中右边析取式中对应的析取支为 $z_p = 1 \wedge y_{r_i} > 0$.

第 4 类约束限制了只有 $x_p$ 不为 0 时 $z_p$ 会大于 0.

第 5 类约束是一个析取式, 左析取支描述进程 $P$ 没有被产生的情况, 相应的规则使用次数也都为 0. 右析取支描述, 若进程 $P$ 被产生, 则至少存在一个能够产生 $P$ 的规则 $r_i$, 其左边的变量 $z_{Y_i}$ 也大于 0.

通过这 5 类约束和一个可达性查询语句, 可以得到输入的 BPP 可达性问题对应的约束集. 这里给出了一个可达性查询输入例子:

initial
*S*
rules
*S* -> *A*
*A* -> *A*, *B*
query
*B* == 1

输入的 BPP 含有两条规则, 初始状态为 $S$. 查询语句 $B$==1 询问符号 $B$ 出现次数为 1 的 BPP 状态是否可达.

通过存在性 Presburger 公式的构造, 输入对应的约束集为

$$\begin{gathered}x_S \geqslant 0,\ x_A \geqslant 0,\ x_B \geqslant 0,\ y_{r_1} \geqslant 0,\ y_{r_2} \geqslant 0,\\ 1+0\cdot y_{r_1}+0\cdot y_{r_2}-y_{r_1}=x_S,\\ 0+1\cdot y_{r_1}+1\cdot y_{r_2}-y_{r_2}=x_A,\\ 0+0\cdot y_{r_1}+1\cdot y_{r_2}=x_B,\\ x_S=0\vee z_S>0,\ x_A=0\vee z_A>0,\ x_B=0\vee z_B>0,\ z_S=0,\\ (z_A=0\wedge y_{r_1}=0\wedge y_{r_2}=0)\vee(z_A=z_S+1\wedge y_{r_1}>0\wedge z_S>0)\vee(z_A=z_A+1\wedge y_{r_2}>0\wedge z_A>0),\\ (z_B=0\wedge y_{r_2}=0)\vee(z_B=z_A+1\wedge y_{r_2}>0\wedge z_A>0),\\ x_B=1.\end{gathered}$$

5.1.3 输 出

输出包含求解结果和求解时间, 用户可以通过指令选择输出构造的约束集和 Z3 求解器提供的统计信息.

求解结果分为两种情况, 若可满足, 则返回模型, 即各变量的一组赋值解; 否则返回导致矛盾的约束集.

以第 5.1.2 节中可达性查询输入示例为例, 工具求解的结果为可满足, 并返回一组赋值$\{(x_S,0),(x_A,1),(x_B,1),(y_{r_1},1),(y_{r_2},1),(z_S,0),(z_A,1),(z_B,2)\}$表示两条规则使用次数都为 1, 与变量 $z_S$, $z_A$ 和 $z_B$ 相关的约束限制了规则的使用顺序: 状态 $S$ 通过规则 $r_1$ 转变为状态 $A$, 再通过规则 $r_2$ 转变为状态$\{A,B\}$, 于是 $S$ 出现 0 次, $A$ 和 $B$ 出现 1 次.

若将查询语句改为 $A$==2, 询问符号 出现次数为 2 的 BPP 状态是否可达, 工具求解的结果是不可满足, 并返回矛盾的约束集: $\{x_B \geqslant 0,\ 1+0\cdot y_{r_1}+0\cdot y_{r_2}-y_{r_1}=x_s,\ 0+1\cdot y_{r_1}+1\cdot y_{r_2}-y_{r_2}=x_A,\ x_A=2\}$. 可知 $A$ 的出现次数与两条规则的使用相关, 又因为 $A$ 出现次数必须为 2, 根据约束 $y_{r_1}$ 的值只能为 2, 于是根据约束 $x_S$ 的值只能为–1, 与约束 $x_S \geqslant 0$ 矛盾, 因此该查询状态不可达.

### 5.2 实验结果

我们提出了基于基本并行进程的异步通信程序验证技术, 在基本并行进程的上近似的语义下, 我们可知若 Actor 通信系统某些状态不可达, 则对应的 Erlang 程序状态也不可达, 从而验证异步通信程序是安全的.

为了评估以上基于基本并行进程的验证方法和 RABLE, 我们对文献[1]提供的几个 Erlang 异步通信程序经典测试用例进行了实验. 各测试用例的内容和验证性质具体如下.

pipe: 验证邮箱的限界性质, 证明了在任意给定时间内, Actor 通信系统中每个进程的邮箱中消息数量至多为 1, 即消息数量≥2 的状态不可达, 从而问题“Actor 通信系统的邮箱 2 限界问题”的答案为否.

ring: 创建高阶并发函数, 递归调用类似 pipe 功能的函数体. 该例验证了异步通信程序邮箱的限界性质.

state_factory: 创建了高阶并发的函数, 通过异步通信对状态进行更新. 该用例验证了邮箱的限界性质的同时, 还验证了任意时间点没有冲突请求同时对状态进行更新, 即互斥性.

reslock 和 reslockbeh: 异步通信的多进程在两种不同实现的锁协议下对共享内存单元进行修改, 通过验证多进程的临界区状态可达性, 验证了程序中互斥锁的正确性.

parikh: 通过判断邮箱中的消息计数是否符合特定要求, 验证了程序中模块被正确初始化的程序安全性.

我们将 RABLE 与当前向量加法系统的高效模型检测工具 BFC 进行了实验比较, BFC 是 Osualdo 等人开发的基于 Actor 通信系统的自动化验证工具 Soter 的后端[1,10].

表 1 给出了对 Actor 通信系统在上述 Erlang 程序上验证的实验结果. 可以看到, 几乎所有用例下通过我们实现的 RABLE 所用的求解时间比 BFC 少; 并且, 通过 Actor 通信系统的基本并行进程语义构造出的规则数量也更少, 因此我们提出的基于基本并行进程的验证方法在验证 Erlang 异步通信程序的性质上更具求解时间和规则数量的优势.

表 1 Actor 通信系统验证的实验结果

| 实验用例 | | BFC | RABLE |
|---|---|---|---|
| pipe | 规则数量 | 13 | 9 |
| | 求解时间 | 0.106 | 0.025 |
| ring | 规则数量 | 39 | 28 |
| | 求解时间 | 0.268 | 0.106 |
| state_factory | 规则数量 | 33 | 22 |
| | 求解时间 | 0.736 | 0.074 |
| reslock | 规则数量 | 51 | 38 |
| | 求解时间 | 0.821 | 0.228 |
| reslockbeh | 规则数量 | 61 | 46 |
| | 求解时间 | 0.856 | 0.268 |
| parikh | 规则数量 | 30 | 20 |
| | 求解时间 | 0.070 | 0.097 |

以上实验展示了我们的方法在验证异步通信程序方面具有一定通用性: 支持如 Erlang 程序的高阶函数、动态创建进程、异步消息通信的行为, “状态可达性问题”能够验证具体错误状态的不可达, 进而验证异步程序中重要的互斥性等性质, 邮箱 $K$ 限界问题能灵活地运用于验证与消息数相关联的具体性质, 如多进程操作执行顺序的有效性、安全性.

最后, 使用我们的方法验证异步通信程序的局限性主要有以下几类: (1) 因程序进行中导致的消息类型错误、算术错误等原因抛出的程序异常无法直接建模, 需要针对性增加额外的抽象或缺陷定位规则; (2) 无法验证依赖通信消息顺序先后的程序安全性, 但实际应用中此类程序比重仍少于当前消息计数器的抽象能够覆盖的多数程序; (3) 与 Actor 通信系统相似, 由于缺乏对栈的抽象, 无法验证依赖栈信息的程序安全性性质.

## 6 总结及未来工作

本文关注异步通信程序的形式化验证问题, 改进了 Osualdo 等人提出的为异步通信程序建模的 Actor 通信系统, 将其归约至基本并行进程. 赋予了基本并行进程的语义, 降低了验证异步通信程序的复杂度, 对 Erlang 程序实例成功进行建模和安全性验证. 我们借助 SMT 求解器开发了模型检测工具 RABLE, 实现了存在性 Presburger 公式这一可达性算法, 能够验证基本并行进程的可达性与限界活性. 使用 RABLE 进行实验, 与已有方法相比在求解时间和约束数量上具有优势.

未来的工作方向以及待解决的问题, 主要包括两个方面: 一是对异步通信程序的具体应用问题进行探索和验证, 如缺陷定位和动态更新等, 希望给出相应问题的形式化定义, 使用基本并行进程的可达性进行建模和验证; 二是继续考虑程序模型, 扩展已有的对通信作近似的思想, 将模型归约到基本并行进程, 例如通过对异步通信下推系统的栈进行限制.

**References**:

[1] Armstrong J. Erlang. CACM, 2010, 53(9): 68−75. [doi: 10.1145/1810891.1810910]

[2] D’Osualdo E, Kochems J, Ong CHL. Automatic verification of erlang-style concurrency. In: Proc. of the Int’l Static Analysis Symp. Berlin: Springer-Verlag, 2013. 454–476.

[3] Carlsson R. An introduction to Core Erlang. In: Proc. of the PLI 2001 Erlang Workshop. 2001.

[4] Kochems J. Verification of asynchronous concurrency and the shaped stack constraint. Oxford: Oxford University, 2014.

[5] Ong CHL, Ramsay SJ. Verifying higher-order functional programs with pattern-matching algebraic data types. In: Proc. of the POPL. 2011. 587−598.

[6] Sen K, Viswanathan M. Model checking multithreaded programs with asynchronous atomic methods. In: Ball T, Jones RB, eds. Lecture Notes in Computer Science: Computer Aided Verification, CAV 2006. Seattle: Springer-Verlag, 2006. 300−314. [doi: 10.1007/11817963_29]

[7] Emmi M, Ganty P, Majumdar R, *et al*. Analysis of asynchronous programs with event-based synchronization. In: Vitek J, ed. Proc. of the 24th European Symp. on Programming, LNCS: Programming Languages and Systems, ESOP 2015. London: Springer-Verlag, 2015. 535−559.

[8] Qadeer S, Rehof J. Context-bounded model checking of concurrent software. In: Halbwachs N, Zuck LD, eds. Lecture Notes in Computer Science: Tools and Algorithms for the Construction and Analysis of Systems, the 11th Int'l Conf. (TACAS). Berlin: Springer-Verlag, 2005. 93−107.

[9] Kochems J, Ong CL. Safety verification of asynchronous pushdown systems with shaped stacks. In: D'Argenio PR, Melgratti HC, eds. Proc. of the CONCUR 2013, Vol. 8052. Berlin: Springer-Verlag, 2013. 288−302.

[10] Leroux J, Schmitz S. Reachability in vector addition systems is primitive recursive in fixed dimension. In: Proc. of the 34th Annual ACM/IEEE Symp. on Logic in Computer Science, LICS 2019. Vancouver: IEEE, 2019. 1−13. [doi: 10.1109/LICS.2019.8785796]

[11] Czerwinski W, Lasota S, Lazic R, *et al*. The reachability problem for Petri nets is not elementar. In: Charikar M, Cohen E, eds. Proc. of the 51st Annual ACM SIGACT Symp. on Theory of Computing, STOC 2019. New York: ACM, 2019. 24–33. [doi: 10.1145/3313276.3316369]

[12] Rackoff C. The covering and boundedness problems for vector addition systems. Theoretical Computer Science 6, 1978, 223–231.

[13] Esparza J. Decidability and complexity of Petri net problems—An introduction. In: Advanced Course on Petri Nets. Berlin: Springer-Verlag, 1996. 374–428. [doi: 10.1007/3-540-65306-6_20]

[14] Kaiser A, Kroening D, Wahl T. Efficient coverability analysis by proof minimization. In: Koutny M, Ulidowski I, eds. Proc. of the CONCUR 2012. LNCS 7454, Berlin: Springer-Verlag, 2012. 500−515. [doi: 10.1007/978-3-642-32940-1_35]

[15] Esparza J, Ledesma-Garza R, Majumdar R, *et al*. An SMT-based approach to coverability analysis. In: Biere A, Bloem R, eds. Proc. of the CAV 2014. Cham: Springer-Verlag, 2014. 603−619.

[16] Christensen S. Decidability and decomposition in process algebras [Ph.D. Thesis]. Edinburgh: The University of Edinburgh, Department of Computer Science, 1993.

[17] Verma KN, Seidl H, Schwentick T. On the complexity of equational horn clauses. In: Nieuwenhuis R, ed. Proc. of the Automated Deduction-CADE2020. LNCS 3632. Berlin: Springer-Verlag, 2005. 337−352.

[18] 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 nglish abstract). http://www.jos.org.cn/1000-9825/5191.htm [doi: 10.13328/j.cnki.jos.005191]

[19] 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]

[20] Tan JH, Li GQ. Bounded model checking liveness on basic parallel processes. Ruan Jian Xue Bao/Journal of Software, 2020, 31(8): 2388−2403 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5959.htm [doi: 10.13328/j.cnki.jos.005959]

[21] Fu H. Model checking EGF on basic parallel processes. In: Proc. of the 9th Int'l Symp. on Automated Technology for Verification and Analysis. Berlin: Springer-Verlag, 2011. 120−134. [doi: 10.1007/978-3-642-24372-1_10]

[22] 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]

[23] Karp RM, Miller RE. Parallel Program Schemata. Journal of Computer and System Sciences, 1969, 3(2): 147−195.

[24] Barrett C, Tinelli C. Satisfiability modulo theories. Journal on Satisfiability Boolean Modeling and Computation, 2018, 3(3): 141–224.

[25] Moura LD, Bjørner N. Z3: An efficient SMT solver. In: Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Berlin: Springer-Verlag, 2008. 337–340.

[26] Tan JH. Verification of asynchronous communication program based on basic parallel processes [MS. Thesis]. Shanghai: Shanghai Jiao Tong University, 2020 (in Chinese with English abstract).

[27] Hague M, Lin A W. Synchronisation- and reversal-bounded analysis of multithreaded programs with counters. In: Madhusudan P, Seshia SA, eds. Proc. of the 24th Int'l Conf. on Computer Aided Verification. LNCS 7358, Berlin: Springer-Verlag, 2012. 260−276.

**附中文参考文献**:

[18] 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]

[19] 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]

[20] 谭锦豪, 李国强. 基本并行进程活性的限界模型检测. 软件学报, 2020, 31(8): 2388−2403. http://www.jos.org.cn/1000-9825/5959.htm [doi: 10.13328/j.cnki.jos.005959]

[26] 谭锦豪. 基于基本并行进程的异步通信程序验证研究 [硕士学位论文]. 上海: 上海交通大学, 2020.

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

**谭锦豪**(1996－), 男, 博士生, 主要研究领域为程序语言理论.

**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.