软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2022,33(8):2918−2929 [doi: 10.13328/j.cnki.jos.006605]

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

基于深度学习和反例制导的循环程序秩函数生成*

林开鹏, 梅国泉, 林 望, 丁佐华

(浙江理工大学 信息学院, 浙江 杭州 310018)

通信作者: 林望, E-mail: linwang@zstu.edu.cn

摘 要: 程序终止性判定是程序分析与验证领域中的一个研究热点. 针对非线性循环程序, 提出了一种基于反例制导的神经网络型秩函数的构造方法. 该方法采用学习组件和验证组件交互的迭代框架, 其中, 学习组件利用程序轨迹作为训练集合构造一个候选秩函数; 验证组件运用可满足性模理论(satisfiability modulo theories, SMT)确保候选秩函数的有效性; 而由 SMT 返回的反例则进一步用于扩展学习组件中的训练集合, 以对候选秩函数进行精化. 实验结果表明, 所提出的方法比已有的机器学习方法在秩函数的构造效率和构造能力上具有优势.

关键词: 秩函数; 反例制导方法; 深度神经网络; 终止性分析; 循环程序

中图法分类号: TP311

中文引用格式: 林开鹏, 梅国泉, 林望, 丁佐华. 基于深度学习和反例制导的循环程序秩函数生成. 软件学报, 2022, 33(8): 2918–2929. http://www.jos.org.cn/1000-9825/6605.htm

英文引用格式: Lin KP, Mei GQ, Lin W, Ding ZH. Ranking Function Synthesis for Loop Programs via Counterexample Guided Deep Learning. Ruan Jian Xue Bao/Journal of Software, 2022, 33(8): 2918−2929 (in Chinese). http://www.jos.org.cn/1000-9825/6605.htm

Ranking Function Synthesis for Loop Programs via Counterexample Guided Deep Learning

LIN Kai-Peng, MEI Guo-Quan, LIN Wang, DING Zuo-Hua

(School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China)

Abstract: This study proposes a novel approach for synthesizing ranking functions that are expressed as feed-forward neural networks. The approach employs a counter example-guided synthesis procedure, where a learner and a verifier interact to synthesize ranking function. The learner trains a candidate ranking function that satisfies the ranking function conditions over a set of sampled data, and the verifier either ensures the validity of the candidate ranking function or yields counterexamples, which are passed back to further guide the learner. The procedure leverages efficient supervised learning algorithm, while guaranteeing formal soundness via SMT solver. The tool SyntheRF is implemented, then, its scalability and effectiveness are evaluated over a set of benchmark examples.

Key words: ranking function; counterexample guided procedure; deep neural networks; termination analysis; loop programs

随着计算机软件应用领域的不断拓宽, 软件系统的可靠性显得至关重要. 作为软件正确性理论的一个重要分支, 程序终止性已引起了人们的广泛关注[1−3]. 对给定的一个程序, 若存在一种输入条件及相应的可执行路径, 使得该程序在此路径上运行而不终止, 则称该程序是不终止的; 否则, 称其为终止的. 显然, 程序的不终止可能会引起计算机系统无响应, 从而导致系统失效. 因此, 程序的终止性研究具有重要的理论意义与应用价值.

研究表明: 尽管程序的终止性验证已被证明是不可判定的, 但对于一些具有特殊结构的循环程序, 其终

* 基金项目: 浙江省自然科学基金(LY20F020020); 上海工业控制系统安全创新功能型平台开放课题; 上海工业控制安全创新科技有限公司资助课题

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: 2021-09-05; 修改时间: 2021-10-14; 采用时间: 2022-01-10; jos 在线出版时间: 2022-01-28

止性是可判定的[4−6]. 例如, 文献[5]基于迁移矩阵特征值分析给出了线性程序终止性判定的充分必要条件. 当前, 常用的方法是借助秩函数(ranking function)来判定循环程序的终止性. 秩函数是一个将程序状态映射到实数集合的函数, 且函数值在进入循环时是有下界的, 且在执行循环体后是严格递减的. 显然, 对给定的循环程序, 若存在一个秩函数, 可表明该程序是终止的.

针对线性秩函数和多项式秩函数的研究已有大量工作. 在线性秩函数生成方面, Colón 和 Sipma 在文献[7]中利用凸多面体理论以及 Farkas 引理给出了一种线性秩函数的计算方法. Podelski 和 Rybalcheko 在文献[8]中针对线性约束的非嵌套循环程序提出了一种线性秩函数的完备构造方法. Amir 与 Samir 等人提出了一种基于线性规划(linear programming)的方法来合成单路径线性约束循环(single-path loops)程序的多阶段秩函数[9]. 根据秩函数的排序方式, Manna 等人在文献[10]中提出了字典秩函数(lexicographic ranking function)的概念, 并利用 Farkas 引理提出了一种线性字典秩函数的计算方法. 此外, 与秩函数排序方式相关的工作还有字典秩函数终止证明[11]、线性分段秩函数(piecewise linear ranking functions)[12]等.

在多项式秩函数生成方面, Chen 等人在文献[13]中, 通过设定循环程序的秩函数模板, 将多项式程序的秩函数合成问题转化为半代数系统求解问题, 然后利用符号计算工具 DISCOVERER 和 QEPCAD 来求解半代数系统, 最终得到秩函数模板中参数的解. 在文献[14]中, Shen 等人将秩函数生成问题转化为多项式优化问题, 并通过半定规划(semidefinite programming, SDP)来计算非线性多项式秩函数. Yuan 等人则将多项式秩函数的生成问题转化为分类问题[15], 并使用支持向量机(support vector machines, SVM)[16,17]生成候选秩函数, 然后运用 SMT 求解器 Z3[18]进行验证. 上述方法均可用于构造多项式循环程序的多项式型秩函数, 但它们都存在各自的缺点. 例如: 基于量词消去(quantifier elimination, QE)的方法较为耗时, 其复杂性被证明是指双数级的[19], 因此不适于大规模的复杂程序; 而基于 SDP 的方法和基于 SVM 的方法求得的是近似的候选秩函数, 因此需要通过 Z3 求解器进行验证. 由于 Z3 求解器无法处理包含超越函数的表达式, 因此当求得的候选秩函数包含超越项时, 该方法无法证明候选秩函数的正确性.

近来, 基于神经网络的技术已被用于程序分析与验证, 如循环程序不变量生成[20,21]、动态系统的稳定性分析[22]等. 基于这种想法, Giacobbe 等人首次提出了基于深度神经网络的循环程序终止性分析方法[23], 该方法训练一个神经网络作为候选秩函数, 并借助 Z3 求解器进行验证. 在文献[24]中, Tan 等人提出了一种深度神经网络型秩函数构造的新方法. 与文献[23]中方法不同, 该方法通过运用优化方法来验证秩函数的正确性, 即不需要借助可满足性模理论(satisfiability modulo theories, SMT)也能证明秩函数的正确性. 上述方法不仅能处理线性与多项式循环程序, 而且能处理带有超越函数的循环程序. 然而, 在运用监督学习方法构造深度神经网络型秩函数时往往需要数目较大的训练样本, 这使得上述方法在秩函数的构造效率方面较受限制.

针对上述问题, 本文提出了一种反例制导的神经网络型秩函数的构造方法. 该方法采用 Learner 组件和 Verifier 组件交互的迭代框架, 其中, Learner 组件利用程序轨迹作为训练集合构造一个候选秩函数, Verifier 组件运用 SMT 求解器 dreal 确保候选秩函数的有效性, 而由 SMT 返回的反例则用于更新 Learner 组件中的训练集合以对候选秩函数进行精化. 重复上述迭代过程, 直至生成一个有效的神经网络型秩函数. 本文提出的方法既利用监督学习方法的高效性, 同时借助 SMT 求解器来保证所求得的秩函数的正确性. 实验结果表明, 所提出的方法比已有的机器学习方法在秩函数的构造效率和表达能力上具有优势.

本文第 1 节介绍了关于循环程序终止性和深度神经网络的相关概念. 第 2 节是主要工作, 详细介绍反例制导的神经网络型秩函数的构造算法. 第 3 节通过一组基准示例对所提出算法进行实验评估, 并与现有的一些方法进行比较. 第 4 节对本文进行总结.

1 预备知识

1.1 循环程序与秩函数

给定循环程序 P, 形式如下:

$$\left.\begin{array}{l}\textbf{while } \bigwedge_{i=1}^{m} g_i(\boldsymbol{x}) \triangleright 0 \textbf{ do}\\ x_1' = f_1(\boldsymbol{x})\\ x_2' = f_2(\boldsymbol{x})\\ \vdots\\ x_n' = f_n(\boldsymbol{x})\end{array}\right\} \tag{1}$$

其中, $\boldsymbol{x}\in\mathbb{R}^n$ 为程序状态向量, $n\in\mathbb{N}$, $\mathbb{R}$为实数; $\triangleright\in\{<,>,\leqslant,\geqslant\}$; $\bigwedge_{i=1}^{m} g_i(\boldsymbol{x}) \triangleright 0$ 表示循环条件; x_1'表示的第 i 维的变量在迭代一次之后的值. 此外, 我们要求循环程序满足以下条件.

(1) 终止的(terminating): 循环程序不能是死循环, 是可终止的;

(2) 确定的(deterministic): 变量的更新是确定的, 即每一次迭代时, 有且仅有一个确定的 $x_i' = f_i(\boldsymbol{x})$.

对于一个如公式(1)所示的循环程序 P, 令 B 为所有满足循环条件的状态集合, 即:

$$B = \left\{ \boldsymbol{x} \middle| \bigwedge_{i=1}^{m} g_i(\boldsymbol{x}) \triangleright 0 \right\} \tag{2}$$

给定一个循环程序 P 以及循环条件 B, 我们给出的秩函数定义如下.

定义 1(秩函数). 给定一个如公式(1)所示的循环程序 P, 如果存在一个函数 $R(\boldsymbol{x}):\mathbb{R}^n\to\mathbb{R}$, 满足以下条件:

(i) $\forall \boldsymbol{x}\in B, R(\boldsymbol{x})-R(\boldsymbol{x}')>c^+$;

(ii) $\forall \boldsymbol{x}\in B, R(\boldsymbol{x})\geqslant 0$,

其中, c^+是一个大于 0 的常数, 那么我们称函数 $R(\boldsymbol{x})$是循环程序 P 的秩函数.

在定义 1 中: 条件(i)说明在执行循环体后, 秩函数 $R(\boldsymbol{x})$的值是严格单调递减的; 根据条件(ii), $R(\boldsymbol{x})$始终是非负的. 因此, 若函数 $R(\boldsymbol{x})$满足上述条件, 则可保证循环程序 P 对任意的初始输入都是终止的[25].

例 1: 给定一个循环程序 P:

$$\begin{array}{l}\textbf{while } (x_1 > 0, x_1 < x_2) \textbf{ do}\\ x_1' = 2x_1\\ x_2' = x_2\end{array}$$

- 当 $R(\boldsymbol{x})=-x_1$ 时, 在该循环程序 P 中, 所有满足循环条件的状态集合为 $B=\{(x_1,x_2)|x_1>0,x_1<x_2\}$. 显然, 函数 $R(\boldsymbol{x})$在该程序中严格递减. 然而, 函数 $R(\boldsymbol{x})$在 B 上的值小于 0, 这违背了秩函数条件(ii), 因此, 函数 $R(\boldsymbol{x})$不是循环程序 P 的秩函数;
- 当 $R(\boldsymbol{x})=x_2-x_1$ 时, 函数 $R(\boldsymbol{x})$随着每次迭代严格递减, 并且下界大于 0, 因此, $R(\boldsymbol{x})$是 P 的秩函数.

1.2 深度神经网络

深度神经网络由许多相互连接的神经元组成, 各个神经元按接收信息的先后分为不同的层, 每一层的神经元接收来自上一层神经元的信号, 并产生信号作为输入传递到下一层. 通常, 输入和输出层的索引分别设为 0 和 L, 输入与输出层之间的中间层称为隐藏层, 其索引设为 l. 当给一个初始输入时, 该输入经过神经网络的输入层, 然后通过连续的线性计算和激活函数在各层中传播, 直到到达输出层.

对于深度神经网络, 非输入层神经元值由前一层的输出、权重矩阵 W 和偏置向量 b 计算得到. 令 x_0 表示输入层的神经元值; z_l 和 x_l, $1\leqslant l\leqslant L$ 分别表示隐藏层 l 中的神经元在经过激活函数 a_l 前、后的值; x_L 表示输出层的神经元值, 则神经网络的前向传播可表示为

$$\begin{cases} z_l = W_l x_{l-1} + b_l, & l = 1,2,...,L-1\\ x_l = a_l(z_l), & l = 1,2,...,L-1\\ x_L = W_L x_{L-1} + b_L \end{cases} \tag{3}$$

根据通用近似定理(universal approximation theorem)[26], 深度神经网络具有很强的拟合能力, 在神经元个

数足够的情况下, 常见的连续非线性函数都可以用深度神经网络来近似.

2 秩函数生成与验证

给定的一个如公式(1)所示的循环程序 P, 本文旨在如何利用深度神经网络构造一个秩函数 $R(\boldsymbol{x})$, 以保证循环程序的终止性. 为此, 我们提出了一个基于深度神经网络和 SMT 求解的反例制导框架. 如图 1 所示, 该框架由 Learner 组件和 Verifier 组件组成. 下面, 我们将分别介绍 Learner 组件和 Verifier 组件.

- Learner 组件: 由循环程序的轨迹构造训练样本集, 并将定义 1 中的秩函数条件编码为损失函数; 然后运用梯度下降方法训练神经网络, 直至损失函数的取值为 0. 此时, 求得的神经网络即为候选的秩函数. 由于该候选秩函数仅在训练集上能满足定义 1 中的秩函数条件, 因此需进一步将该候选秩函数传递给 Verifier 组件进行验证, 以保证该候选秩函数的正确性;
- Verifier 组件: 对于求得的一个候选秩函数, 运用 SMT 求解器进行验证. 如果候选秩函数满足定义 1 中的秩函数条件, 则说明该候选秩函数是正确的; 否则, SMT 求解器返回一个不满足秩函数条件的反例, 并利用该反例对训练集进行更新, 以重新训练新的候选秩函数.

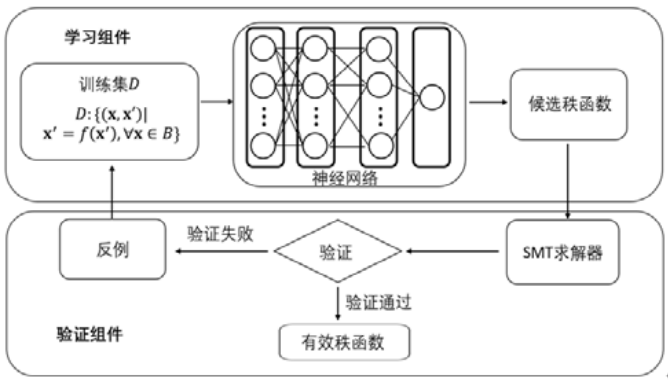

图 1 面向秩函数生成的反例制导架构

在上述反例制导框架中, Learner 组件和 Verifier 组件交替迭代运行, 直至 Verifier 组件返回一个有效的秩函数, 或者达到预设的最大迭代次数.

2.1 候选秩函数生成

在 Learner 组件中, 我们以前馈神经网络为模板来构造候选秩函数. 前馈神经网络的结构如下.

- 一个输入层、输出层, L–1 个隐藏层;
- 输入层的维数等于循环程序中状态变量的个数;
- 输出层的维度为 1, 即输出的结果为一个标量.

对于神经网络, 激活函数在学习和理解非线性函数时有着非常重要的作用. 激活函数是连接上一层到下一层神经元值的非线性函数, 因此增强了神经网络的表示能力和学习能力. 常用的激活函数有 *sigmoid* 型激活函数、*Relu* 激活函数、*Swish* 激活函数等. 在文献[23,24]中, *Relu* 函数和 *sigmoid* 函数分别作为合法的激活函数, 其中,

$$sigmoid(x)=\frac{1}{1+\mathrm{e}^{-x}},\ Relu(x)=\begin{cases}x, & x>0\\ 0, & x\leqslant 0\end{cases}.$$

为了更好地呈现本文方法的有效性, 我们也选取 *Relu* 函数和 *sigmoid* 函数作为神经网络的激活函数.

2.1.1 训练集的构造

在 Learner 组件中, 我们利用循环程序的轨迹来构造深度学习所需的训练样本集合 X_{train}. 具体地, 给定一

个如公式(1)所示的循环程序 P, 其满足循环条件的状态空间 B 如公式(2)所示, 训练样本集 X_{train} 的构造方式则可表示为

$$X_{train}:=proj(filter(mesh(\boldsymbol{l},\boldsymbol{u}))),$$

其中, *mesh* 和 *filter* 分别表示网格构建和过滤过程.

首先, 我们运用 $mesh[l,u]$函数对状态空间 B 进行网格采样. 令 $\boldsymbol{l},\boldsymbol{u}\in\mathbb{R}^n$ 分别为状态空间 B 的上、下界. 对超矩形$[\boldsymbol{l},\boldsymbol{u}]$的各维度, 以预定的网格间距 s 进行等距采样可得到格点集合 M; 然后, 我们对格点集合 M 进行 *filter* 运算, 即依据程序 P 的循环条件对集合 M 中的各格点进行筛选, 去除不满足循环条件的格点, 从而得到神经网络训练时所需的样本数据集 $X_{train}:=M\cap B$. 显然, 当 B 本身即为超矩形$[\boldsymbol{l},\boldsymbol{u}]$时, $X_{train}=M$. 需要说明的是: 当循环程序的状态空间 B 为无界区域时, 我们借鉴了文献[24]中的方法, 将状态空间 B 映射到一个有界的状态空间, 并在该有界状态空间中进行网格采样, 具体实现请见文献[24].

2.1.2 损失函数

在深度学习中, 通常借助损失函数来估量神经网络模型的预测值与真实值之间的差异. 损失函数是一个非负实数函数. 令θ为网络参数, 包含所有的权重与偏置. 在 Learner 组件中, 我们试通过不断地更新网络参数 θ, 从而使得所得到的神经网络模型能满足秩函数的条件. 为此, 我们将定义 1 中的秩函数条件编码为深度学习中的损失函数, 以刻画神经网络模型违反秩函数条件的程度. 下面, 我们将介绍如何根据秩函数条件设置损失函数.

首先, 为保证定义 1 中的秩函数条件(i): $\forall\boldsymbol{x}\in B, R(\boldsymbol{x})-R(\boldsymbol{x}')>c^+$成立, 我们引入损失函数:

$$\mathcal{L}_1=\mathbb{E}_{x\sim\rho_B(B)}(\max\{-R(\boldsymbol{x})+R(\boldsymbol{x}')+c^+,0\}) \tag{4}$$

其中, $\boldsymbol{x}'$是 $\boldsymbol{x}$ 在执行循环体之后的值, 即 $\boldsymbol{x}'=f(\boldsymbol{x})$; c^+是大于 0 的常数. 对任意的 $x\in B$ 使得 $R(\boldsymbol{x})-R(\boldsymbol{x}')<c^+$, 损失函数 L_1 对其施加惩罚, 从而使得 $R(\boldsymbol{x})-R(\boldsymbol{x}')$在 B 上趋于正值. 显然, 有:

$$\forall\boldsymbol{x}\in B, (\mathcal{L}_1=0\Rightarrow R(\boldsymbol{x})-R(\boldsymbol{x}')\geqslant c^+).$$

成立, 即当$\mathcal{L}_1=0$ 时, 候选秩函数 $R(\boldsymbol{x})$满足$\forall\boldsymbol{x}\in B, R(\boldsymbol{x})-R(\boldsymbol{x}')\geqslant c^+$.

同样地, 为保证定义 1 中的秩函数条件(ii): $\forall\boldsymbol{x}\in B, R(\boldsymbol{x})\geqslant 0$ 成立, 我们引入损失函数:

$$\mathcal{L}_2=\mathbb{E}_{x\sim\rho_B(B)}(\max\{-R(\boldsymbol{x}),0\}) \tag{5}$$

对任意的 $x\in B$ 使得 $R(\boldsymbol{x})<0$, 损失函数$\mathcal{L}_2$ 对其施加惩罚, 从而使得 $R(\boldsymbol{x})$在 B 上趋于非负. 显然, 有:

$$\forall\boldsymbol{x}\in B, (\mathcal{L}_2=0\Rightarrow R(\boldsymbol{x})\geqslant 0) \tag{6}$$

成立, 即当$\mathcal{L}_2=0$ 时, 候选秩函数 $R(\boldsymbol{x})$满足$\forall\boldsymbol{x}\in B, R(\boldsymbol{x})\geqslant 0$.

综上所述, 秩函数生成问题可以转化为损失函数$\mathcal{L}(\theta)=\mathcal{L}_1+\mathcal{L}_2$ 的最小化问题. 显然, 损失函数$\mathcal{L}(\theta)$是非负的. 对任意的一组网络参数θ^*, 使得$\mathcal{L}(\theta^*)=0$, 与之相应的候选秩函数必然满足定义 1 中的秩函数条件, 因此是一个有效的秩函数. 反之, 有效的秩函数则定义了损失函数$\mathcal{L}(\theta)$的全局最优解. 也就是说, 对于任意的有效秩函数, 与之相应的网络参数θ^*满足$\mathcal{L}(\theta^*)=0$, 即θ^*为$\mathcal{L}(\theta)$的一个全局最优解.

然而, 由于不知道真实程序状态在状态空间 B 上的分布, 在实际中, 我们无法计算期望损失$\mathcal{L}(\theta)$. 因此, 根据经验风险最小化准则, 我们试计算一组网络参数θ^*使得经验损失 $\tilde{\mathcal{L}}(\theta)$:

$$\tilde{\mathbb{L}}(\theta)=\sum_{i=1}^{n}\max\{-R(\boldsymbol{x})+R(\boldsymbol{x}')+c^+,0\}+\sum_{i=1}^{n}\max\{-R(x),0\} \tag{7}$$

在训练集合 X_{train} 上最小, 即 $\theta^*=\arg\min_{\theta}\tilde{\mathcal{L}}(\theta)$. 显然, 当存在一组网络参数$\theta^*$使得 $\tilde{\mathcal{L}}(\theta^*)=0$, 与之相应的候选秩函数 $R(\boldsymbol{x})$在训练集 X_{train} 上满足定义 1 中的秩函数条件. 本文采用经典的 Adam 优化器, 通过梯度下降算法实现经验损失函数最小化, 从而得到相应的候选秩函数.

例 2: 给定一个循环程序 P, 表达式为

$$\textbf{while } x_1^2+x_2^2+x_3^2\leqslant 1 \textbf{ do}$$
$$x_1'=x_1-1-\cos(x_3)$$
$$x_2'=x_2-1-\cos(x_1)$$
$$x_3'=x_3-1-\sin(x_3)$$

试构造一个秩函数 R, 使得 $\forall x\in\{x_1,x_2,x_3\mid x_1^2+x_2^2+x_3^2\leqslant 1\}, R(x)-R(x')\geq c^+\wedge R(x)\geqslant 0$ 成立.

构造一个 3 层的神经网络, 其中, 输入层、隐藏层与输出层节点数分别为 3, 10, 1, 选择 *sigmoid* 函数作为激活函数, 并用学习率为 0.001 的 Adam 优化器来训练这个神经网络, 我们可以得到一个候选秩函数 $R(x)$, 满足以下条件:

$$\begin{cases} z_1=W_1\boldsymbol{x}_1+b_1, \boldsymbol{x}_1=[x_1,x_2,x_3]^{\mathrm{T}} \\ \boldsymbol{x}_2=sigmoid(z_1) \\ R(x)=W_2\boldsymbol{x}_2+b_2 \end{cases} \tag{8}$$

其中,

$$W_1=\begin{bmatrix} -0.58851 & -0.55490 & -0.39862 & -0.31029 & 0.14352 & 0.36681 & 0.75939 & 1.02156 & 1.06512 & 1.55494 \\ -1.22047 & 0.51520 & -2.10911 & 0.69139 & -0.81984 & -0.56088 & -0.16346 & 1.04451 & 2.62539 & 0.31080 \\ -1.01813 & -0.62673 & -1.63820 & 1.36523 & 0.34825 & -0.02534 & -1.42174 & -0.26419 & 1.58296 & -0.54971 \end{bmatrix}^{\mathrm{T}},$$

$b_1=[0.00188\ 0.00123\ 0.00404\ 0.00219\ 0.00224\ 0.00266\ 0.00669\ 0.00268\ 0.00407\ 0.00887]^{\mathrm{T}}$,

$W_2=[0.93271\ 0.93348\ 0.93183\ 1.07352\ 0.95441\ 0.93963\ 0.92702\ 1.06152\ 1.06726\ ]$,

$b_2=[0.23429]$.

在训练过程中, 经验损失函数与迭代次数的关系如图 2 所示. 可以看出: 经验损失随着迭代次数的增加而迅速下降, 并在 580 次迭代后收敛到 0. 这说明上述的候选秩函数 $R(\boldsymbol{x})$能保证在训练集上满足秩函数条件, 而无法保证在状态空间 B 上也满足秩函数条件.

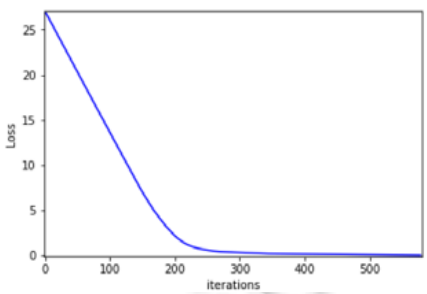

图 2 候选秩函数损失值随着迭代快速减少

2.2 秩函数验证与反例生成

如前所述, 在 Learner 组件中, 通过学习神经网络参数, 使得经验损失函数最小化的方法仅保证所求得的候选秩函数在训练集上满足秩函数条件. 因此, 我们还需借助 Verifier 组件对 Learner 组件计算得到的候选秩函数进行进一步验证, 以确保候选秩函数的正确性.

在 Verifier 组件中, 我们使用了 SMT 求解器 dreal 来验证该候选秩函数是否满足定义 1 中的秩函数条件. 若不满足, SMT 求解器则返回一个违反秩函数条件的状态作为反例. 根据秩函数的定义, 若一个候选秩函数满足下列条件:

$$\forall \boldsymbol{x}\in B, R(\boldsymbol{x})\geqslant 0\wedge R(\boldsymbol{x})-R(\boldsymbol{x}')\geqslant c^+ \tag{9}$$

则称其为循环程序 P 的秩函数. 这可进一步转化为验证条件(9)的逆反命题:

$$\exists \boldsymbol{x}\in B, (R(\boldsymbol{x})<0)\vee R(\boldsymbol{x})-R(\boldsymbol{x}')<c^+ \tag{10}$$

若公式(10)存在解, 则该解可作为候选秩函数关于秩函数条件(10)的反例, 并进一步用于扩充 Learner 组

件中的训练集, 而提高下一轮迭代训练的精确性. 相反, 若公式(10)经 SMT 求解器判定为不可满足的, 即找不到状态 $\boldsymbol{x}$ 使得公式(10)成立, 则说明候选秩函数能严格满足秩函数条件(10), 因此是循环程序 P 的有效秩函数.

在上述反例制导的迭代学习框架中, 反例的选取是优化候选秩函数的关键. 基于这一特征, 我们力图一次搜索多个反例, 进而用于引导 Learner 组件改进候选秩函数. 通常, SMT 求解器在求解时难以同时生产多个反例, 幸运的是, 根据函数的连续性, 反例周围的样本点同样可能违反秩函数条件. 因此, 我们在反例的邻域中进行随机采样, 进而根据条件(10)进行确认, 并将确认为反例的样本添加至下一轮迭代的训练集. 上述方式有效减少了迭代学习的时间, 从而提高了训练候选秩函数的效率.

例 2 续: 对于公式(8)中的候选秩函数 $R(\boldsymbol{x})$, 我们使用 SMT 求解器 dreal 找到了一个反例 $\boldsymbol{x}$:(x_1=0.7192, x_2=−0.6689, x_3=−0.1870), 该反例违反了秩函数条件 $R(\boldsymbol{x})-R(\boldsymbol{x}')<c^+$. 采用上述的基于反例制导的迭代学习框架, 经过 4 次迭代后, 我们得到了循环程序 P 的一个有效秩函数 $R(\boldsymbol{x})$:

$$\begin{cases} z_1 = W_1\boldsymbol{x}_1 + b_1, \boldsymbol{x}_1 = [x_1, x_2, x_3]^{\mathrm{T}} \\ \boldsymbol{x}_2 = sigmoid(z_1) \\ R(x) = W_2\boldsymbol{x}_2 + b_2 \end{cases}$$

其中,

$$W_1 = \begin{bmatrix} 0.92969 & -0.97519 & 0.56415 & 0.16971 & 1.42025 & 0.61081 & -1.40393 & -1.9905 & 1.42759 & -1.78298 \\ 0.79271 & -1.25771 & -1.16492 & 0.16850 & 1.23613 & 0.06772 & 1.40757 & 0.32609 & 0.10745 & 0.35209 \\ 0.41964 & 0.50886 & -0.11197 & 0.03039 & -0.71144 & 0.51805 & -0.97699 & 0.64773 & -0.40218 & 0.69932 \end{bmatrix}^{\mathrm{T}},$$

$$b_1 = [0.06188\ 0.06123\ 0.06404\ 0.09709\ 0.06221\ 0.07246\ 0.06069\ 0.07234\ 0.07207\ 0.07387]^{\mathrm{T}},$$

$$W_2 = [1.05307\ 0.94812\ 0.94271\ 1.074532\ 1.05045\ 1.06237\ 0.94162\ 0.94101\ 1.05940\ 0.93969\ ],$$

$$b_2 = [0.13123].$$

2.3 秩函数生成算法

算法 1 中描述了基于反例制导的秩函数生成的主要实现步骤. 该算法以循环程序 P、最大迭代次数 *maxIter* 作为输入, 以秩函数为输出. Learner 组件和 Verifier 组件分别由算法 1 中的 *LEARNER* 和 *VERIFIER* 函数实现.

算法 1.

输入: 一个循环程序 P 如公式(1), 最大迭代次数 *maxIter*;

输出: 有效秩函数 R.

```
1:  LEARNER(X_train)
2:      repeat
3:          R←NN_θ   //NN_θ代表神经网络, R 代表候选秩函数, θ为神经网络参数
4:          计算神经网络损失函数 L̃(θ)
5:          θ ← θ + α∇_θ L̃
6:      until 损失函数收敛到 0
7:      return R
8:  VERIFIER(R,P)
9:      验证候选秩函数是否满足条件(10)
10:     反例 or 不满足←SMT 求解器(R,P)
11:     return 反例 or 不满足
12: MAIN(·)
13:     X_train←对 P 采样
14:     初始化 N_θ
15:     while iter<maxIter do
```

16: $N_\theta \leftarrow LEARNING(X\_data)$

17: **switch** $VERIFICATION(N_\theta, P)$ **do**

18: case 不满足: return N_θ //验证结果为不满足说明 N_θ为真秩函数.

19: case 反例: $X_data \leftarrow X_data \cup$反例

我们首先构建训练数据集 X_{train}, 并初始化神经网络(第 13 行、第 14 行). 第 15 行~第 19 行是该算法的主要执行步骤. 函数 *LEARNER* 通过使用梯度下降优化方法更新网络参数θ来最小化经验损失函数(第 4 行、第 5 行), 当损失函数收敛至 0 时返回一个候选秩函数, 并将其作为 *VERIFIER* 函数的输入. *VERIFIER* 函数通过 SMT 求解器验证秩函数条件的可满足性(第 9 行、第 10 行). 当返回结果为不满足时, 说明该候选秩函数为有效秩函数, 并将其作为结果输出, 程序结束; 否则生成反例, 并将其用于扩充下一次迭代中的训练数据集. 重复上述步骤, 直到达到最大迭代次数 *maxIter*.

需要指出的是, 算法 1 并不能提供完备的终止性判定结果. 如前所述, 对给定的循环程序, 算法 1 若能找到相应的秩函数, 则表明该程序是终止的. 然而, 当算法 1 在给定的最大迭代次数内仍不能求得秩函数时, 我们无法给出任何关于循环程序终止性的分析结果. 图 3 列举了两个选自文献[24,27]的非终止循环程序. 对于非终止循环程序 1, 运用算法 1 得到了一个候选秩函数, 然而经 SMT 求解器验证, 该候选秩函数存在违反秩函数条件的反例, 并且在迭代次数超过 10 次后仍是如此. 因此, 我们判定算法 1 无法求得有效的秩函数.

```
1  int s = 1, t = 1, c = 1
2  while(t*t – 4*s + 2*t + 1 + c >=0):
3      t = t + 2
4      s = s + t
5      c = c + t
```

非终止循环程序 1

```
1  int x = 0, y = 0, z=-1.2
2  while(x*x + y*y <=1 , z<=0):
3      x = 1/4 * x + 1/8 * y
4      y = 1/4 * x*x
5      z = 1/6 * z – 1
```

非终止循环程序 2

图 3 非终止循环程序

对于非终止循环程序 2, 如图 4 所示, 在运用算法 1 计算秩函数时, 出现经验损失函数在训练集上无法收敛到 0 的问题. 因此, 我们判定算法 1 无法求得候选值函数.

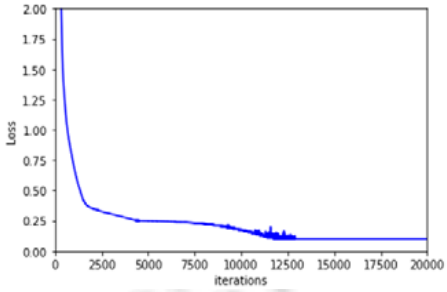

图 4 非终止循环程序 2 的经验损失函数在训练集上无法收敛到 0

3 实验结果及分析

基于算法 1, 我们借助 tensorflow 平台实现了相应的原型工具 SyntheRF. 文中所有的实验均在配有 64GB RAM, 3.7 GHz Intel Core-i9 处理器上进行. 在实验中, 我们选取 3 层结构的神经网络模型, 输入节点数量依据程序变量的维度而定, 隐藏层节点数设为 10, 输出层节点数设为 1. 此外, Learner 组件中的优化器学习率设为 0.01, c^+设为 0.01, 并选取 *sigmoid*、*Relu* 函数为激活函数选; 选取 dreal 工具作为 Verifier 组件中的 SMT 求解器, 其中, dreal 参数δ设置为 0.000 1, 验证时间上限设置为 1 000 s.

3.1 实验案例来源

附录 A 中给出了文中实验的所有实例, 其中, 循环程序 1、循环程序 11、循环程序 12、循环程序 15 选自

文献[15], 循环程序 16 来自文献[9], 其他循环程序来自文献[24]. 这些实例既包含了维度不同的循环程序, 又包含了多项式循环程序和带有超越函数的循环程序等.

3.2 实验结果

表 1 给出了 SyntheRF 与文献[24]中的 TAN-LI 方法的实验结果. 由于文献[24]中没有给出 TAN-LI 方法生成秩函数所需的具体时间, 为此, 我们复现了该方法, 从而给出了每个例子的实验时间. 在与 TAN-LI 方法对比时, 我们使用了与之相同的 sigmoid 激活函数.

表 1 SyntheRF 与 TAN-LI 方法结果对比

Examples	Network structure	SyntheRF				TAN-LI 方法[24]	
		t_l	t_v	*Samples*	*Iters*	*t*	*Samples*
1	1-10-1	0.30	0.09	20	1	2.87	221
2	1-10-1	0.17	0.08	20	1	0.25	17
3	1-10-1	0.25	0.01	22	1	0.40	227
4	1-10-1	0.24	0.01	22	1	0.27	227
5	2-10-1	0.09	0.06	55	2	40.58	28 483
6	2-10-1	0.17	0.01	110	1	0.22	5 223
7	2-10-1	0.30	0.01	313	1	1.13	43 541
8	2-10-1	0.68	0.01	313	2	18.48	43 541
9	2-10-1	0.71	0.11	313	1	0.89	43 541
10	2-10-1	0.26	0.05	110	1	8.42	9 045
11	2-10-1	0.39	0.04	313	2	17.43	43 541
12	2-10-1	0.14	0.08	150	2	30.81	27 167
13	2-10-1	0.23	2.59	901	2	12.26	53 361
14	2-10-1	0.15	0.90	901	2	11.05	53 908
15	2-10-1	0.12	0.02	15	1	0.34	602
16	2-10-1	4.72	0.01	1 225	2	14.59	9 045
17	2-10-1	0.45	0.01	820	1	1.03	9 045
18	2-10-1	0.18	0.05	1 225	2	72.11	62 500
19	3-10-1	0.09	0.02	110	1	2.38	6 786
20	3-10-1	0.10	1.21	4 160	4	456.03	452 865
21	3-10-1	0.11	5.95	4 160	3	26.52	229 919
22	3-10-1	0.10	0.07	2 060	1	9.62	143 100

在表 1 中: 编号与附录中所列实例相一致, 网络结构表示各实例中采用的神经网络模型的具体结构, t_l 表示 Learner 组件计算候选秩函数所需的神经网络训练时间, t_v 表示 Verifier 组件判定候选秩函数正确性所需的验证时间, *Samples* 表示初始训练集的大小, *Iters* 表示反例制导迭代学习的训练次数. 在“TAN-LI 方法”中, *t* 和 *Samples* 的含义与“SyntheRF”中的相同. 由于 TAN-LI 方法采用优化方法判定候选秩函数的正确性, 所需的验证时间相对较少, 这里忽略不计.

从表 1 可以看出: SyntheRF 和 TAN-LI 方法均成功合成了所有 22 个循环程序的秩函数, 但就初始训练集规模而言, SyntheRF 所需的训练样本数量较 TAN-LI 方法更少, 尤其对于编号 5、编号 7–编号 9、编号 20、编号 21 等对应的循环程序, 相差更是明显. 这是因为 TAN-LI 方法为保证深度学习方法计算得到的候选秩函数能严格满足秩函数条件而对采样间隔有严格要求, 但 SyntheRF 不需要固定的采样间隔, 可根据具体循环程序进行灵活采样. 此外, 在计算候选秩函数所需的神经网络训练时间上, SyntheRF 所花费的时间也少于 TAN-LI 方法, 例如编号 5、编号 12、编号 18、编号 20、编号 21 等对应的循环程序. 总的来说, SyntheRF 平均需 1.78s 生成有效的秩函数, 而 TAN-LI 方法平均需要 60s. 与此同时, SyntheRF 所需初始训练集样本数目仅为 TAN-LI 方法的 2%. 由此可见, SyntheRF 较 TAN-LI 方法更为有效.

对于一些没有借助反例增强的例子, 如表 1 中编号为 7, 9, 10, 19 对应的循环程序, 仅经过一次学习-验证过程就得到一个有效的秩函数. 这说明对应的循环程序秩函数的解空间较大, 使用较少的训练集也能够生成有效的秩函数, 因此在训练时间上相较略少.

此外, 对于表 1 中编号 5、编号 12、编号 18、编号 20 等对应的循环程序, 由于 SyntheRF 采用了反例制导的深度学习方法实现秩函数生成, 其所花费的训练时间远少于 TAN-LI 方法, 这能较好地反映了反例制导方

法的作用.

表 2 展示了 SyntheRF 与文献[22]中的 Giacobbe 方法以及文献[15]中基于支持向量机(SVM)的方法的实验评估结果. 特别地, 在与 Giacobbe 方法相比较时, 我们使用了与之相同的 *Relu* 激活函数. 在表 2 中, T 代表成功找到有效秩函数, F 代表未能找到有效秩函数.

表 2 SyntheRF 与 Giacobbe 方法、基于 SVM 的方法的结果对比

编号	Giacobbe 方法[23]	基于 SVM 的方法[15]	SyntheRF	编号	Giacobbe 方法[23]	基于 SVM 的方法[15]	SyntheRF
1	T	T	T	12	T	T	T
2	T	F	T	13	T	T	T
3	T	F	T	14	T	T	T
4	T	F	T	15	T	T	T
5	T	T	T	16	F	F	T
6	T	T	T	17	T	T	T
7	F	F	T	18	T	F	T
8	F	F	T	19	F	F	T
9	F	F	T	20	F	F	T
10	T	F	T	21	F	F	T
11	F	T	T	22	T	T	T

在文献[22]中, Giacobbe 等人使用了 *Relu* 函数作为神经网络的激活函数, 并将隐藏层与输出层之间的权重设为 1, 从而保证所合成的候选秩函数始终是非负的. 由于这种神经网络模型的特殊结构, 使得 Giacobbe 方法在处理编号 7–编号 9、编号 11、编号 16、编号 19–编号 21 等对应的循环程序的秩函数生成问题时失效, 而 SyntheRF 则均能成功生成有效的秩函数.

此外, 文献[15]中基于 SVM 的方法需借助 Z3 求解器验证候选秩函数的正确性, 而 Z3 求解器无法处理具有超越项的逻辑公式. 因此, 基于 SVM 的方法无法处理编号 3、编号 4、编号 7–编号 10、编号 19–编号 21 对应的含有超越函数的循环程序. 而 SyntheRF 不仅可以处理多项式循环程序, 也可以处理含有超越项的循环程序.

综上所述, SyntheRF 较已有的基于机器学习的秩函数生成方法相比, 在构造效率和构造能力上具有一定优势.

4 总 结

本文提出了一种基于反例制导合成非线性秩函数的方法. 该方法采用迭代的方式, 主要通过 Learner 组件与 Verifier 组件生成有效的秩函数, 其中, Learner 组件训练深度神经网络生成候选秩函数, Verifier 组件通过 SMT 求解器验证秩函数条件. 两者相互迭代, 直至生成有效的秩函数. 实验结果表明, 我们提出的方法比已有的机器学习方法在秩函数构造效率和构造能力上具有优势.

References:

[1] Cousot P, Cousot R. An abstract interpretation framework for termination. ACM SIGPLAN Notices, 2012, 47(1): 245–258.

[2] Lee CS, Jones ND, Ben-Amram AM. The size-change principle for program termination. In: Proc. of the 28th ACM SIGPLAN-SIGACT Symp. on Principles of Programming Languages. 2001. 81–92.

[3] Urban C. The abstract domain of segmented ranking functions. In: Proc. of the Int'l Static Analysis Symp. Berlin, Heidelberg: Springer, 2013. 43–62.

[4] Braverman M. Termination of integer linear programs. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer, 2006. 372–385.

[5] Tiwari A. Termination of linear programs. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer, 2004. 70–82.

[6] Numbers ONC. With an application to the entschenidugsproblem. 1937.

[7] Colón MA, Sipma HB. Practical methods for proving program termination. In: Proc. of the Int'l Conf. on Computer Aided Verification. Berlin, Heidelberg: Springer, 2002. 442–454.

[8] Podelski A, Rybalchenko A. A complete method for the synthesis of linear ranking functions. In: Proc. of the Int'l Workshop on Verification, Model Checking, and Abstract Interpretation. Berlin, Heidelberg: Springer, 2004. 239−251.

[9] Ben-Amram AM, Genaim S. On multiphase-linear ranking functions. In: Proc. of the Int'l Conf. on Computer Aided Verification. Cham: Springer, 2017: 601−620.

[10] Bradley AR, Manna Z, Sipma HB. The polyranking principle. In: Proc. of the Int'l Colloquium on Automata, Languages, and Programming. Berlin, Heidelberg: Springer, 2005. 1349−1361.

[11] Cook B, See A, Zuleger F. Ramsey vs. lexicographic termination proving. In: Proc. of the Int'l Conf. on Tools and Algorithms for the Construction and Analysis of Systems. Berlin, Heidelberg: Springer, 2013. 47−61.

[12] Urban C. The abstract domain of segmented ranking functions. In: Proc. of the Int'l Static Analysis Symp. Berlin, Heidelberg: Springer, 2013. 43−62.

[13] Chen Y, Xia B, Yang L, *et al*. Discovering non-linear ranking functions by solving semi-algebraic systems. In: Proc. of the Int'l Colloquium on Theoretical Aspects of Computing. Berlin, Heidelberg: Springer, 2007. 34−49.

[14] Shen L, Wu M, Yang Z, *et al*. Generating exact nonlinear ranking functions by symbolic-numeric hybrid method. Journal of Systems Science and Complexity, 2013, 26(2): 291−301.

[15] Yuan Y, Li Y. Ranking function detection via SVM: A more general method. Proc. of the IEEE Access, 2019, 7: 9971−9979.

[16] Fan RE, Chang KW, Hsieh CJ, *et al*. LIBLINEAR: A library for large linear classification. Proc. of the Journal of Machine Learning Research, 2008, 9: 1871−1874.

[17] Li Y, Sun X, Li Y, *et al*. Synthesizing nested ranking functions for loop programs via SVM. In: Proc. of the Int'l Conf. on Formal Engineering Methods. Cham: Springer, 2019. 438−454. https://doi.org/10.1007/978-3-030-32409-4_27

[18] De Moura L, 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, Heidelberg: Springer, 2008. 337−340.

[19] Brown CW, Davenport JH. The complexity of quantifier elimination and cylindrical algebraic decomposition. In: Proc. of the 2007 Int'l Symp. on Symbolic and Algebraic Computation (ISSAC 2007). New York: Association for Computing Machinery, 2007. 54−60.

[20] Si X, Dai H, Raghothaman M, *et al*. Learning loop invariants for program verification. In: Proc. of the of Neural Information Processing Systems. 2018. 7762−7773.

[21] Chang YC, Roohi N, Gao S. Neural Lyapunov control. In: Proc. of the 33rd Int'l Conf. on Neural Information Processing Systems. 2019. 3245−3254.

[22] Abate A, Ahmed D, Giacobbe M, *et al*. Formal synthesis of Lyapunov neural networks. Proc. of the IEEE Control Systems Letters, 2020, 5(3): 773−778.

[23] Giacobbe M, Kroening D, Parsert J. Neural termination analysis. arXiv preprint arXiv: 2102.03824, 2021.

[24] Tan W, Li Y. Synthesis of ranking functions via DNN. Proc. of the Neural Computing and Applications, 2021: 1−21. https://doi.org/10.1007/s00521-021-05763-8

[25] Floyd RW. Assigning meanings to programs. In: Proc. of the Symp. in Applied llathematics. 1967.

[26] Cybenko G. Approximations by superpositions of a sigmoidal function. Proc. of the Mathematics of Control, Signals and Systems, 1989, 2: 183−192.

[27] Le TC, Antonopoulos T, Fathololumi P, *et al*. DynamiTe: Dynamic termination and non-termination proofs. Proc. of the ACM on Programming Languages, 2020, 4(OOPSLA): 1−30.

附录 A

表 3 循环例子

#	维度	Loop programs
1	1	**while** $x\geqslant 1, x\leqslant 3$ **do** $x'=5x-x^2$
2	1	**while** $x>4$ **do** $x'=-2x+4$
3	1	**while** $\sin(x)\geqslant 0\ x\geqslant 1, x\leqslant 6$ **do** $x'=-x$
4	1	**while** $\sin(x)\geqslant 0\ x\geqslant 1, x\leqslant 6$ **do** $x'=x+1+\cos(x)^2$
5	2	**while** $x_1-x_2\geqslant 1,\ x_1+x_2\geqslant 1,\ x_1\leqslant 2$ **do** $x_1'=x_1-1,\ x_2'=x_2-1$
6	2	**while** $x_1\geqslant 0,\ x_2+2x_1\geqslant 1,\ x_2\leqslant 3$ **do** $x_1'=-x_1^2-4x_2^2+1,\ x_2'=-x_1x_2-1$
7	2	**while** $x_1^2+x_2^2\leqslant 1,\ \cos(x_1)\geqslant 0$ **do** $x_1'=x_1^2+1,\ x_2'=x_2^2-1$

表 3 循环例子(续)

#	维度	Loop programs
8	2	**while** $x_1^2+x_2^2\leqslant 1$ **do** $x_1'=x_1-1-\cos(x_1),\ x_2'=x_2-1-\cos(x_2)$
9	2	**while** $x_1^2+x_2^2\leqslant 1,\ \cos(x_1)\geqslant 0$ **do** $x_1'=x_1-1-\sin(x_1)^2,\ x_2'=x_2-1-\cos(x_2)^3$
10	2	**while** $x_1\geqslant 1,\ x_1\leqslant x_2$ **do** $x_1'=\frac{1}{16}x_1+2+\mathrm{e}^{-x_1},\ x_2'=\frac{1}{16}x_2+1$
11	2	**while** $x_1^2+x_2^2\leqslant 1$ **do** $x_1'=x_1-x_2^2+1,\ x_2'=x_2+x_1^2-1$
12	2	**while** $x_1+x_2\geqslant 1,\ x_1\leqslant 3,\ x_2^2\leqslant 1$ **do** $x_1'=5x_1-x_1^2,\ x_2'=x_2^2+x_2$
13	2	**while** $x_1^2\leqslant 1,\ x_2^2\leqslant 1$ **do** $x_1'=x_1+x_2,\ x_2'=x_2-1$
14	2	**while** $x_2^2-x_2+1\leqslant x_1$ **do** $x_1'=-x_1,\ x_2'=-x_2$
15	2	**while** $x_1\geqslant 1,\ x_2^2+2x_1\leqslant 3x_2$ **do** $x_1'=1+\frac{1}{x_1^2},\ x_2'=-x_1x_2-3x_2+x_2^2+1$
16	2	**while** $x_1>0,\ x_1\leqslant x_2$ **do** $x_1'=2x_1,\ x_2'=x_2+1$
17	2	**while** $x_1>1,\ x_1\leqslant x_2$ **do** $x_1'=x_1+x_2,\ x_2'=-x_2$
18	2	**while** $x_1>0,\ x_2\geqslant 0$ **do** $x_1'=x_1+x_2,\ x_2'=x_2-1$
19	3	**while** $x_2>0,\ x_2\leqslant x_1,\ x_1\leqslant 1$ **do** $x_1'=x_1+1,\ x_2'=\frac{1}{4}\tan(x_2)+1,\ x_3'=-x_3$
20	3	**while** $x_1^2+x_2^2+x_3^3\leqslant 1$ **do** $x_1'=x_1-1-\cos(x_1),\ x_2'=x_2-1-\cos(x_1),\ x_3'=x_3-1-\sin(x_3)$
21	3	**while** $x_1^2+x_2^2+x_3^3\leqslant 1,\ \sin(x_2)\geqslant 0$ **do** $x_1'=x_1-1,\ x_2'=x_2,\ x_3'=\sin(x_3)^3$
22	3	**while** $x_2\geqslant 1,\ x_2\leqslant x_1,\ x_3\geqslant 0$ **do** $x_1'=-x_1,\ x_2'=x_2,\ x_3'=x_3+1$

林开鹏(1997－), 男, 硕士生, CCF 学生会员, 主要研究领域为形式化方法.

林望(1982－), 男, 博士, 副教授, CCF 专业会员, 主要研究领域为形式化方法, 软件分析与验证.

梅国泉(2001－), 男, 主要研究领域为形式化方法.

丁佐华(1964－), 男, 博士, 教授, 博士生导师, CCF 高级会员, 主要研究领域为智能系统软件建模, 分析与测试.