Deep Specification Mining

深度规格挖掘

Tien-Duy B. Le

Tien-Duy B. Le

School of Information Systems

信息系统学院

Singapore Management University, Singapore btdle.2012@smu.edu.sg

新加坡管理大学，新加坡

ABSTRACT

摘要

Formal specifications are essential but usually unavailable in soft-ware systems.Furthermore, writing these specifications is costly and requires skills from developers.Recently, many automated techniques have been proposed to mine specifications in various formats including finite-state automaton (FSA).However, more works in specification mining are needed to further improve the accuracy of the inferred specifications.

正式规范是必要的，但通常在软件系统中不可用。此外，编写这些规范成本很高，需要开发人员的技能。最近，许多自动化技术被提出来挖掘各种格式的规范，包括有限状态自动机(FSA)。然而，规范挖掘还需要更多的工作来进一步提高推断规范的准确性。

In this work, we propose Deep Specification Miner (DSM), a new approach that performs deep learning for mining FSA-based specifications.Our proposed approach uses test case generation to generate a richer set of execution traces for training a Recurrent Neural Network Based Language Model (RNNLM).From these execution traces, we construct a Prefix Tree Acceptor (PTA) and use the learned RNNLM to extract many features.These features are subsequently utilized by clustering algorithms to merge similar automata states in the PTA for constructing a number of FSAs.Then, our approach performs a model selection heuristic to estimate F-measure of FSAs and returns the one with the highest estimated F-measure.We execute DSM to mine specifications of 11 target library classes.Our empirical analysis shows that DSM achieves an average F-measure of 71.97%, outperforming the best performing baseline by 28.22%.We also demonstrate the value of DSM in sandboxing Android apps.

在这项工作中，我们提出了深度规范挖掘器(DSM)，这是一种为挖掘基于 FSA 的规范执行深度学习的新方法。我们提出的方法使用测试用例生成来生成更丰富的执行轨迹集，用于训练基于递归神经网络的语言模型。从这些执行轨迹中，我们构造了一个前缀树接受器(PTA)，并使用所学习的 RNNLM 来提取许多特征。这些特征随后被聚类算法用来合并 PTA 中相似的自动机状态，以构建多个 FSa。然后，我们的方法执行一个模型选择启发式算法来估计金融服务协议的金融服务度量，并返回具有最高估计金融服务度量的金融服务协议。我们执行需求侧管理来挖掘 11 个目标库类的规范。我们的实证分析表明，需求侧管理的平均离差为 71.97%，比最佳绩效基线高出 28.22%。我们还展示了帝斯曼在沙盒安卓应用中的价值。

CCS CONCEPTS

CCS 概念

• Software and its engineering → Dynamic analysis;

软件及其工程→动态分析；

KEYWORDS

关键词

Specification Mining, Deep Learning

规范挖掘，深度学习

ACM Reference Format:

ACM 参考格式:

Tien-Duy B. Le and David Lo.2018.Deep Specification Mining.In Pro-ceedings of 27th ACM SIGSOFT International Symposium on Software Test-ing and Analysis (ISSTA'18).ACM, New York, NY, USA, 12 pages.https: //doi.org/10.1145/3213846.3213876

天都乐和卢大伟。2018.深度规格挖掘。在第 27 届美国计算机学会软件测试与分析国际研讨会(ISSTA'18)的代表中。美国纽约州纽约市 ACM，12 页。https://doi . org/10 . 1145/3213846 . 3213876

1 INTRODUCTION

1 简介

Due to rapid evolution to meet demands of clients, software ap-plications and libraries are often released without documented

由于满足客户需求的快速发展，软件应用程序和库经常在没有文档记录的情况下发布

David Lo

卢大伟

School of Information Systems

信息系统学院

Singapore Management University, Singapore davidlo@smu.edu.sg

新加坡管理大学，新加坡大卫路

specifications.Even when formal specifications are available, they may become outdated as software systems quickly evolve [51] in a short period of time.Finally, writing formal specifications requires necessary skill and motivation from developers, as this is a costly and time consuming process [23].Furthermore, the lack of speci-fications negatively impacts the maintainability and reliability of systems.With no documented specifications, developers may find it difficult to comprehend a piece of code and software is more likely to have bugs due to mistaken assumptions.Furthermore, developers cannot utilize state-of-the-art bug finding and testing tools that need formal specifications as an input [10, 37].

规格。即使有正式的规范，随着软件系统在短时间内快速发展，它们也可能会过时。最后，编写正式的规范需要开发人员具备必要的技能和动力，因为这是一个昂贵且耗时的过程，[23]。此外，缺乏规范会对系统的可维护性和可靠性产生负面影响。由于没有文档化的规范，开发人员可能会发现很难理解一段代码，并且由于错误的假设，软件更有可能有错误。此外，开发人员不能利用需要正式规范作为输入的最先进的错误发现和测试工具[10，37]。

Recently, many automated approaches have been proposed to help developers reduce the cost of manually drafting formal speci-fications [6, 13, 24, 26, 33].In this work, we focus on the family of specification mining algorithms that infer finite-state automaton (FSA) based specifications from execution traces.Krka et al. [24] and many other researchers have proposed various FSA-mining approaches that have improved the quality of inferred FSA models as compared to prior solutions.Nevertheless, the quality of mined specifications is not perfect yet, and more works need to be done to make specification mining better.In fact, FSA based specification miners still suffer from many issues.For instance, if methods in input execution traces frequently occur in a particular order or the amount of input traces is too small, FSAs inferred by k-tails [7] and many other algorithms are likely to return FSAs that are not generalized and overfitted to the input execution traces.

最近，许多自动化方法被提出来帮助开发人员降低手工起草正式规范的成本，[6，13，24，26，33]。在这项工作中，我们着重于规范挖掘算法家族，这些算法从执行轨迹中推断基于有限状态自动机的规范。克尔卡等人，[24]和许多其他研究人员提出了各种金融服务分析挖掘方法，与以前的解决方案相比，这些方法提高了推断的金融服务分析模型的质量。然而，挖掘出的规范的质量还不完善，需要做更多的工作来使规范挖掘更好。事实上，基于 FSA 的规范挖掘者仍然面临许多问题。例如，如果输入执行迹线中的方法经常以特定顺序出现，或者输入迹线的数量太小，则由 k-tails [7]和许多其他算法推断的 FSA 可能会返回未被一般化和过度写入输入执行迹线的 FSA。

To mine more accurate FSA models, we propose a new speci-fication mining algorithm that performs deep learning on execu-tion traces.We name our approach DSM which stands for Deep Specification Miner.Our approach takes as input a target library class C and employs an automated test case generation tool to gen-erate thousands of test cases.The goal of this test case generation process is to capture a richer set of valid sequences of invoked methods of C. Next, we perform deep learning on execution traces of generated test cases to train a Recurrent Neural Network Lan-guage Model (RNNLM) [39].After this step, we construct a Prefix Tree Acceptor (PTA) from the

execution traces and leverage the learned language model to extract a number of interesting features from PTA's nodes.These features are then input to clustering al-gorithms for merging similar states (i.e., PTA's nodes).The output of an application of a clustering algorithm is a simpler and more generalized FSA that reflects the training execution traces.Finally, our approach predicts the accuracy of constructed FSAs (generated by different clustering algorithms considering different settings) and outputs the one with highest predicted value of F-measure.

为了挖掘更精确的 FSA 模型，我们提出了一种新的对执行轨迹进行深度学习的特定挖掘算法。我们将我们的方法命名为 DSM，它代表深度规格矿工。我们的方法将目标库类 C 作为输入，并使用自动化测试用例生成工具生成数千个测试用例。这个测试用例生成过程的目标是捕获一组更丰富的调用方法的有效序列。接下来，我们对生成的测试用例的执行轨迹进行深度学习，以训练递归神经网络语言模型(RNNLM)[39]。在这一步之后，我们从执行轨迹中构造一个前缀树接受者(PTA)，并利用所学的语言模型从 PTA 的节点中提取一些有趣的特征。然后，这些特征被输入到聚类算法中，用于合并相似的状态(即 PTA 的节点)。聚类算法应用程序的输出是反映训练执行轨迹的更简单和更一般化的 FSA。最后，我们的方法预测了构造的模糊聚类算法(由考虑不同设置的不同聚类算法生成)的准确性，并输出了具有最高预测值的模糊聚类算法。

We evaluate our proposed approach for 11 target library classes which were used before to evaluate many prior work [24, 26].For each of the input class, we first run Randoop to generate thousands of test cases.Then, we use execution traces generated by running

我们评估了我们为 11 个目标库类提出的方法，这些目标库类以前用于评估许多以前的工作[24，26]。对于每个输入类，我们首先运行 Randoop 来生成数千个测试用例。然后，我们使用通过运行生成的执行跟踪

106

these test cases to infer FSAs.Our experiments show that DSM achieves an average F-measure of 71.97%.Compared to other exist-ing specification mining algorithms, our approach outperforms all baselines that construct FSAs from execution traces (e.g., k-tails [7], SEKT [24], TEMI [24], etc.) by at least 28.22%.Some of the baselines first use Daikon to learn invariants that are then used to infer a better FSA.Our approach does not use Daikon invariants in the inference of FSAs.Excluding baselines that use Daikon invariants, our approach can outperform the remaining best performing miner by 33.24% in terms of average F-measure.

这些测试用例来推断 FSAs。实验表明，需求侧管理的平均离差为 71.97%。与其他现有的规范挖掘算法相比，我们的方法优于所有根据执行轨迹构建 FSAs 的基线(例如，k-tails [7、SEKT [24、TEMI [24 等)。)至少 28.22%。一些基线首先使用 Daikon 来学习不变量，然后用来推断更好的 FSA。我们的方法在 FSAs 的推理中没有使用 Daikon 不变量。排除使用 Daikon 不变量的基线，我们的方法在平均 F 度量方面可以比剩余的表现最好的挖掘器高出 33.24%。

Additionally, we assess the applicability of FSAs mined by DSM in detecting malicious behaviors in Android apps.We propose a technique that leverages a FSA output by DSM mining algorithm as a behavior model to construct an Android sandbox.Our technique outputs a comprehensive sandbox that considers execution context of sensitive API methods to better protect app users.Our compar-ative evaluation finds that our technique can increase the True Positive Rate of Boxmate [21], a

state-of-the-art sandbox mining approach, by 15.69%, while only increasing False Positive Rate by 4.52%.Replacing DSM with the best performing applicable baseline results in a sandbox that can achieve a similar True Positive Rate (as DSM) but substantially worse False Positive Rate (i.e., False Positive Rate increases by close to 10%).The results indicate it is promising to employ FSAs mined by DSM to create more effective Android sandboxes.

此外，我们还评估了 DSM 挖掘的 FSA 在检测安卓应用中恶意行为的适用性。我们提出了一种利用 DSM 挖掘算法输出的 FSA 作为行为模型来构建安卓沙箱的技术。我们的技术输出了一个全面的沙箱，它考虑了敏感的应用程序接口方法的执行上下文，以更好地保护应用程序用户。我们的对比评估发现，我们的技术可以将最先进的沙箱挖掘方法 Boxsate[21 的真阳性率提高 15.69%，而仅将假阳性率提高 4.52%。用性能最佳的适用基线替换需求侧管理，可以获得相似的真实阳性率(如需求侧管理)，但假阳性率明显更低(即假阳性率增加近 10%)。结果表明，利用帝斯曼挖掘的金融服务协议来创建更有效的安卓沙盒是有希望的。

The contributions of our work are highlighted below: (1) We propose DSM (Deep Specification Miner), a new specifica-

我们工作的贡献如下:(1)我们提出了深度规格挖掘，一种新的规格

tion mining algorithm that utilizes test case generation, deep learning, clustering, and model selection strategy to infer FSA based specifications.To the best of our knowledge, we are the first to use deep learning for mining specifications.

利用测试用例生成、深度学习、聚类和模型选择策略来推断基于 FSA 的规范的操作挖掘算法。据我们所知，我们是第一个将深度学习用于采矿规范的人。

(2) We evaluate the effectiveness of DSM on 11 different target library classes.Our results show that our approach outperforms the best baseline by a substantial margin in terms of average F-measure.

(2)我们评估了需求侧管理在 11 个不同目标图书馆类别中的有效性。我们的结果表明，我们的方法在平均离差度量方面远远优于最佳基线。

(3) We propose a technique that employs a FSA inferred by DSM to construct a more comprehensive sandbox that considers execution context of sensitive API methods.Our evaluation shows that our proposed technique can outperform several baselines by a substantial margin in terms of either True Positive Rate or False Positive Rate.

(3)我们提出了一种技术，该技术使用由需求侧管理推断的有限状态机来构建一个更全面的沙箱，该沙箱考虑了敏感应用编程接口方法的执行上下文。我们的评估表明，无论是真阳性率还是假阳性率，我们提出的技术都可以大大超过几个基线。

The remainder of this paper is structured as follows.Section 2 highlights background materials.Sections 3 and 4 present DSM and its evaluation.Section 5 presents our proposed technique that employs a FSA inferred by DSM for detecting malicious behaviors in Android apps, along with its evaluation.We discuss threats to validity and related works in Section 6 and Section 7, respectively.Finally, we conclude and mention future work in Section 8.

本文的其余部分结构如下。第 2 节重点介绍背景材料。第 3 节和第 4 节介绍需求侧管理及其评估。第 5 节介绍了我们提出的技术，该技术使用 DSM 推断的 FSA 来检测安卓应用程序中的恶意行为，并对其进行评估。我们分别在第 6 节和第 7 节讨论有效性威胁和相关工作。最后，我们在第 8 节中总结并提及未来的工作。

2 BACKGROUND

2 背景

Statistical Language Model: A statistical language model is an oracle that can foresee how likely a sentence s = w1,w2, ...,wn to occur in a language.In a nutshell, a statistical language model considers a sequence s to be a list of words w1,w2, ...,wn and as-signs probability to s by computing joint probability of words: P(w1, ...,wn) = În−i=1 P(wi |w1, ...,wi−1).As it is challenging to

统计语言模型:统计语言模型是预言一个句子 s = w1，w2，.。。，wn 出现在一种语言中。简而言之，统计语言模型认为序列 s 是单词 w1、w2，...通过计算单词的联合概率:P(w1，.。。，wn)=μn I = 1 P(wi | w1，.。。，wi-1)。因为挑战在于

STN

STN

NT

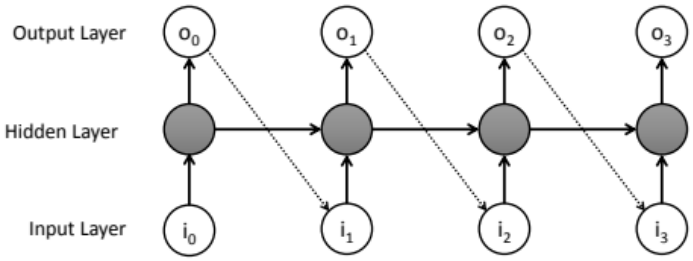
新界

HMTF

HMTF

<END>

<结束>



<START>

<开始>

STN

STN

NT

新界

HMTF

HMTF

t0 t1 t2 t3 time

t0 t1 t2 t3 时间

Figure 1: An unrolled Recurrent Neural Network from time t0 to t3 for predicting the next likely method given a se-quence of invoked methods for java.util.StringTokenizer. "STN" : StringTokenizer(), "NT" : nextToken(), and "HMTF" : hasMoreTokens()==false.

图 1:一个从时间 t0 到 t3 展开的递归神经网络，用于预测下一个可能的方法，给出了调用 java.util.StringTokenizer 的方法序列。

compute conditional probability P(wi |w1, ...,wi−1), each differ-ent language model has its own assumption to approximate the calculation.N-grams model, a popular family of language mod-els, approximates in a way that a word wk conditionally depends only on its previous N words (i.e., wk−N +1 , ...,wk−1).For exam-ple, unigram model simply estimates P(wi |w1, ...,wi−1) as P(wi), bigram model approximates P(wi |w1, ...,wi−1) as P(wi |wi−1), etc.In this work, we utilize the ability of language models to compute P(wi |w1, ...,wi−1) for estimating features of automaton states.We consider every method invocation as a word and an execution trace of an object as a sentence (i.e., sequence of method invocations).Given a sequence of previously invoked methods, we use a language model to output the probability of a method to be invoked next.

计算条件概率 ρ(wi | w1，.。。每个不同的语言模型都有自己的假设来近似计算。N-gram 模型是一个流行的语言模型家族，其近似方式是一个单词 wk 有条件地只依赖于它之前的 N 个单词(即 wk-N+1，.。。，wk 1)。例如，单图模型简单地估计了 P(wi |w1，.。。二元模型近似为 P(wi | w1，.。。，wi-1)作为 P(wi | wi-1)等。在这项工作中，我们利用语言模型的能力来计算 P(wi | w1，.。。wi-1)用于估计自动机状态的特征。我们认为每个方法调用都是一个单词，对象的执行轨迹是一个句子(即方法调用的序列)。给定一系列先前调用的方法，我们使用语言模型输出下一个调用方法的概率。

Recurrent Neural Network Based Language Model: Recently, a family of language models that make use of neural networks is shown to be more effective than n-grams [38].These models are referred to as neural network based language models (NNLM).If a NNLM has many hidden layers, we refer to the model as a deep neural network language model or deep language model for short.Among these deep language models, Recurrent Neural Network Based Language Model (RNNLM) [39] is well-known with its ability to use internal memories to handle sequences of words with arbi-trary lengths.The underlying network architecture of a RNNLM is a Recurrent Neural Network (RNN) that stores information of input word sequences in its hidden layers.Figure 1 demonstrates how a RNN operates given the sequence <START>, STN, NT, HMTF, <END>.In the figure, a RNN is unrolled to become four connected networks, each of which is processing one input method at a time step.Initially, all states in the hidden layer are assigned to zeros.At time tk , a method mk is represented as an one-hot vector ik by the input layer.Next, the hidden layer updates its states by using the vectorik and the states previously computed at time tk−1 .Then, the output layer estimates a probability vector ok across all methods for them to appear in the next time step tk+1 .This process is repeated at subsequent time steps until the last method in the sequence is handled.

基于递归神经网络的语言模型:最近，一个使用神经网络的语言模型家族被证明比 n-gram[38]更有效。这些模型被称为基于神经网络的语言模型(NNLM)。如果一个 NNLM 有许多隐藏层，我们称之为深层神经网络语言模型或简称深层语言模型。在这些深层语言模型中，基于递归神经网络的语言模型(RNNLM)[39]以其使用内部记忆处理任意长度单词序列的能力而闻名。RNNLM 的底层网络结构是递归神经网络(RNN)，它在其隐藏层中存储输入单词序列的信息。图 1 展示了给定序列<开始>、

STN、NT、HMTF、<结束>，RNN 是如何操作的。在图中，一个 RNN 展开成四个相连的网络，每个网络一次处理一种输入法。最初，隐藏层中的所有状态都被分配为零。在时间 tk，方法 mk 由输入层表示为单热向量 ik。接下来，隐藏层通过使用矢量和先前在时间 tk-1 计算的状态来更新其状态。然后，输出层估计所有方法的概率向量 ok，以便它们出现在下一个时间步骤 tk+1 中。在随后的时间步骤中重复该过程，直到处理完序列中的最后一个方法。

107

3 PROPOSED APPROACH

3 提议的方法

Figure 2 shows the overall framework of our proposed approach.In our framework, there are three major processes: test case gener-ation and traces collection, Recurrent Neural Network Based Language Model (RNNLM) learning, and automata construction.Our approach takes as input a target class and signatures of methods.Then, DSM runs Randoop [42] to generate a substantial number of test cases for the input target class.Then, we record the execution of these test cases, and retain traces of invocations of methods of the input target class as the training dataset.Next, our approach performs deep learning on the collected traces to infer a RNNLM that is capable of predicting the next likely method to be executed given a sequence of previously called methods.We choose RNNLM over traditional probabilistic language models since past studies show its superiority [39, 44].

图 2 显示了我们提议的方法的总体框架。在我们的框架中，有三个主要过程:测试用例生成和跟踪收集、基于递归神经网络的语言模型学习和自动机构建。我们的方法将目标类和方法签名作为输入。然后，DSM 运行 Randoop [42]，为输入目标类生成大量测试用例。然后，我们记录这些测试用例的执行，并保留输入目标类的方法调用的跟踪作为训练数据集。接下来，我们的方法对收集到的轨迹进行深度学习，以推断出一个 RNNLM，该 RNNLM 能够在给定一系列先前调用的方法的情况下预测下一个可能执行的方法。我们选择 RNNLM 而不是传统的概率语言模型，因为过去的研究表明它的优越性[39，44]。

Subsequently, we employ a heuristic to select a subset of traces that best represents the whole training dataset.From these traces, we construct a Prefix Tree Acceptor (PTA);we refer to each PTA's node as an automaton state.We select the subset of traces in order to optimize the performance when constructing PTA, but still maintaining accuracy of inferred FSAs.Utilizing the inferred RNNLM, we extract a number of features from automaton states, and input the feature values to a number of clustering algorithms (i.e., k-means [35] and hierarchical clustering [43]) considering different settings (e.g., different number of clusters).The output of a clustering algorithm are clusters of similar automaton states.We use these clusters to create a new FSA by merging states that belong to the same cluster.Every application of a clustering algorithm with a particular setting results in a different FSA.We propose a model selection strategy to heuristically select the most accurate model by predicting values of Precision, Recall, and F-measure.Finally, we output the FSA with highest predicted F-measure.

随后，我们采用启发式方法来选择最能代表整个训练数据集的轨迹子集。根据这些轨迹，我们构建了前缀树接受者；我们将每个 PTA 的节点称为自动机状态。我们选择迹线的子集，以便在构建 PTA 时优化性能，但仍然保持推断的 FSa 的准确性。利用推断的 RNNLM，我们从自动机状态中提取多个特征，并

将特征值输入考虑不同设置(例如，不同数量的聚类)的多个聚类算法(即，k-means [35]和分层聚类[43])。聚类算法的输出是相似自动机状态的聚类。我们使用这些集群通过合并属于同一集群的状态来创建新的 FSA。具有特定设置的聚类算法的每个应用都会导致不同的 FSA。我们提出了一种模型选择策略，通过预测精度、召回率和离差来启发式地选择最准确的模型。最后，我们输出具有最高预测 f 测度的金融稳定分析。

3.1 Test Case Generation and Trace Collection

3.1 测试用例生成和跟踪收集

This process plays an important role to our approach as it decides the quality of RNNLM inferred by the deep learning process.Previ-ous research works in specification mining [24, 26, 27] collect traces from the execution of a program given unit test cases or inputs manually created by researchers.In this work, we utilize deep learn-ing for mining specification.Deep learning requires a substantially large and rich amount of data.The more training inputs, the more patterns the resultant RNNLM can capture.In general, it is difficult to follow previous works to collect a rich enough set of execution traces for an arbitrary target library class.Firstly, it is challenging to look for all projects that use the target library class, especially for classes from new or unreleased libraries.Secondly, existing unit test cases or manually created inputs may not cover many of the possible execution scenarios of methods in a target class.

这一过程对我们的方法起着重要的作用，因为它决定了由深度学习过程推断的 RNNLM 的质量。先前在规范挖掘[24，26，27]中的研究工作收集了给定单元测试用例或研究者手工创建的输入的程序执行的轨迹。在这项工作中，我们利用对挖掘规范的深入学习。深度学习需要大量丰富的数据。训练输入越多，生成的 RNNLM 可以捕获的模式就越多。一般来说，很难按照以前的工作为任意目标库类收集足够丰富的执行跟踪集。首先，寻找所有使用目标库类的项目很有挑战性，尤其是来自新的或未发布的库的类。其次，现有的单元测试用例或手动创建的输入可能无法涵盖目标类中方法的许多可能的执行场景。

We address the above issues by following Dallmeier et al. [11, 12] to generate as many test cases as possible for mining specifications, and collect the execution traces of these test cases for subsequent steps.Recently, many test case generation tools have been proposed such as Randoop[42], EvoSuite[16], etc.Among the state-of-the-art test case generation tools, we choose Randoop because

我们通过遵循 Dallmeier 等人的[11，12]来解决上述问题，以便为挖掘规范生成尽可能多的测试用例，并为后续步骤收集这些测试用例的执行轨迹。最近，已经提出了许多测试用例生成工具，例如兰多普[42、埃沃套件[16 等。在最先进的测试用例生成工具中，我们选择 Randoop 是因为

https://randoop.github.io/randoop/ http://www.evosuite.org/

https://randoop.github.io/randoop/·http://www.evosuite.org/

it is widely used and lightweight.Furthermore, Randoop is well maintained and frequently updated with new versions.As future work, we plan to integrate many other test case generation methods into our approach.

它用途广泛，重量轻。此外，Randoop 维护良好，并经常更新新版本。作为未来的工作，我们计划将许多其他测试用例生成方法集成到我们的方法中。

Randoop generates a large number of test cases, which is pro-portional to the time limit of its execution.In order to improve the coverage of possible sequences of methods under test, we pro-vide class-specific literals aside from default ones to Randoop.For example, for java.net.Socket, we create string and integer liter-als which are addresses of hosts (e.g., "localhost", "127.0.0.1", etc.)

and listening ports (e.g., 8888, etc.).Furthermore, we create dri-ver classes that contain static methods that invoke constructors of the target class to initialize new objects.That helps speed up Randoop to create new objects without spending time to search for appropriate input values for constructors.

Randoop 生成大量测试用例，这与其执行的时间限制成比例。为了提高测试中可能的方法序列的覆盖率，除了缺省值之外，我们还向 Randoop 提供了特定于类的文本。例如，对于 java.net.Socket，我们创建字符串和整数升，它们是主机的地址(例如，“localhost”、“127.0.0.1”等)。)和监听端口(例如 8888 等)。)。此外，我们创建包含静态方法的 dri-ver 类，这些静态方法调用目标类的构造函数来初始化新对象。这有助于加快 Randoop 创建新对象的速度，而无需花费时间为构造函数搜索合适的输入值。

3.2 Learning RNNLM for Specification Mining

3.2 学习面向规范挖掘的 RNNLM

3.2.1 Construction of Training Method Sequences.Our set of collected execution traces is a series of method sequences.Each of these sequences starts and ends with two special symbols: <START> and <END>, respectively.These symbols are used for separating two different sequences.We gather all sequences together to create data for training Recurrent Neural Networks.Furthermore, we limit the maximum frequency of a method sequence MAX_SEQ_FREQ to 10 to prevent imbalanced data issue where a sequence appears much more frequently than the other ones.

3.2.1 训练方法序列的构建。我们收集的执行跟踪集是一系列方法序列。每个序列都以两个特殊符号开始和结束:<开始>和<结束>。这些符号用于分隔两个不同的序列。我们收集所有序列来创建用于训练递归神经网络的数据。此外，我们将方法序列 MAX_SEQ_FREQ 的最大频率限制为 10，以防止出现不平衡的数据问题，其中一个序列比其他序列出现得更频繁。

3.2.2 Model Training.We perform deep learning on the train-ing data to learn a Recurrent Neural Network Based Language Model (RNNLM) for every target library class.By default, we use Long Short-Term Memory (LSTM) network [20], one of the state-of-the-art RNNs, as the underlying architecture of the RNNLM.Compared to the standard RNN architecture, LSTM is better in learning long-term dependencies.Furthermore, LSTM is scalable for long sequences [44].

3.2.2 模型培训。我们对训练数据进行深度学习，以便为每个目标库类学习基于递归神经网络的语言模型。默认情况下，我们使用长短期内存(LSTM)网络[20 作为 RNNLM 的底层架构，这是最先进的 RNN 之一。与标准的 RNN 架构相比，LSTM 在学习长期依赖性方面更好。此外，LSTM 对于长序列是可伸缩的[44]。

3.3 Automata Construction

3.3 自动机构造

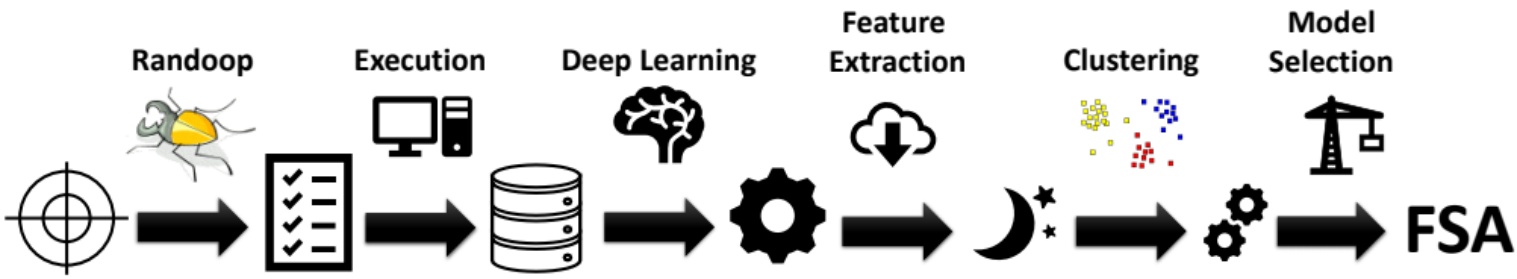
In this processing step, our approach takes as input the set of train-ing execution traces and the inferred RNNLM (see Section 3.2).The output of this step is a FSA that best captures the specification of the corresponding target class.The construction of FSA undergoes several substeps: trace sampling, feature extraction, clustering, and model selection.

在这个处理步骤中，我们的方法将训练执行轨迹集和推断的 RNNLM 作为输入(参见第 3.2 节)。这个步骤的输出是一个 FSA，它最好地捕获了相应目标类的规范。FSA 的构建经历了几个子步骤:跟踪采样、特征提取、聚类和模型选择。

At first, we use a heuristic to select a subset of method sequences that represents all training execution traces.The feature extrac-tion and clustering steps use these selected traces, instead of all traces, to reduce computation cost.We construct a Prefix Tree Ac-ceptor (PTA) from the selected traces and extract features for every PTA nodes using the inferred RNNLM.We refer to each PTA node as an automaton state.Figure 3 shows an excerpt of an example PTA constructed from sequences of invocations of methods from java.security.Signature. Our goal is to find similar automaton states and group them into one cluster.In the clustering substep, we run a number of clustering algorithms on PTA nodes with various settings to create many different FSAs.Finally, in the model selec-tion substep, we follow a heuristic to predict the F-measure (see Section 4.2.1) of constructed FSAs and output the one with highest

首先，我们使用启发式方法来选择代表所有训练执行轨迹的方法序列子集。特征提取和聚类步骤使用这些选择的轨迹，而不是所有轨迹，以降低计算成本。我们从所选择的轨迹中构造一个前缀树接受器，并使用推断的神经网络模型提取每个前缀树接受器节点的特征。我们将每个 PTA 节点称为自动机状态。图 3 显示了从 java.security.Signature 的方法调用序列中构建的一个示例 PTA 的摘录。在聚类子步骤中，我们在具有不同设置的 PTA 节点上运行许多聚类算法来创建许多不同的 FSa。最后，在模型选择子步骤中，我们遵循一种启发式方法来预测构造的金融服务协议的金融服务度量(见第 4.2.1 节)，并输出具有最高值的金融服务协议

108



Target Class & Methods

目标类和方法

Test Cases

测试用例

Traces

跟踪

RNN Based

总部在 RNN

Language Model

语言模型

Features

特征

FSA

FSA

Candidates

候选人

Figure 2: DSM's Overall Framework

图 2:需求侧管理的总体框架

Algorithm 1: Selecting Subset of Execution Traces

算法 1:选择执行跟踪的子集

Input :S = {Si | 1 ≤ i ≤ N}: Collection of execution traces where N is the number of traces
Output :O: Selected execution traces 1 Sort S in ascending order of trace length 2 D ← initialization of dictionary type 3 P ← empty set 4 for Si ∈ S do

输入:S = {Si | 1 ≤ i ≤ N}:执行跟踪的集合，其中 N 是跟踪的数量输出:O:选定的执行跟踪 1 按跟踪长度的升序排序 S 2d←初始化字典类型 3 P ←为 Si ∈ S do 空集 4

5 for (a, b) where a ∈ Si ∧ b ∈ Si ∧ a < b do 6 D[(a, b)]+=[Si] 7 Include (a, b) to P 8 end 9 end

5 代表(a，b)，其中 a∈Si∈b∈Si∈a < b 做 6 维[(a，b)]+=[Si] 7 包括(a，b)到 P 8 末端 9 末端

S9

S9

S1

S1

<START>

<开始>

S2

S2

<init> S6

<初始化> S6

update

更新

S7 S8

S7 S8

initVerify <END>

初始验证<结束>

S10 S12

S10 S12

initVerify <END>

初始验证<结束>

S11

S11

initVerify

初始验证

S3 S4 S5

S3·S4 S5

verify <END>

验证<结束>

… …

… …

Figure 3: An Example Prefix Tree Acceptor (PTA) include Si to O. We mark the pair (a, b) as processed by removing it from P (line 15).We keep searching for sequences until O covers all co-occurrence pairs (a, b) in the input execution traces.

图 3:前缀树接受器的例子包括从硅到氧。我们通过从磷(第 15 行)中移除来标记对(a，b)。我们一直在搜索序列，直到 0 覆盖输入执行轨迹中的所有同现对(a，b)。

Table 1: Extracted Features for An Automata State

表 1:自动机状态的提取特征

10 O ← empty set 11 while ∃(a, b) ∈ P do

10 O ←空套 11 而∃(a，b←p 做

12 Select (a, b) ∈ P such that D[(a, b)] has the least number of elements

12 选择(a，b) ∈ P，使[(a，b)]的元素数量最少

13 Find Si ∈ D[(a, b)] such that Si is shortest ∧ Si < O 14 Include Si to O and remove all pairs (a, b) from P where

13 找到 Si ∈ D[(a，b)]，使 Si 最短∈Si < o14 包括 Si 到 O，并从 P 中删除所有对(a，b)，其中

a ∈ Si ∧ b ∈ Si ∧ a < b 15 end

a∈Si∈b∈Si∈a < b15 端

16 return O

16 返回 0

predicted F-measure.The full set of traces is used in this model selection step.In the following paragraphs, we describe details of each substep in this processing step:

预测离差。该模型选择步骤中使用了全套轨迹。在下面的段落中，我们将描述该处理步骤中每个子步骤的细节:

Trace Sampling: Our training data contains a large number of sequences.Thus, it is expensive to use all of them for constructing FSAs.Therefore, the goal of trace sampling is to create a smaller subset that is likely to represent the whole set of all traces reason-ably well.We propose a heuristic to find a subset of traces that covers all co-occurrence pairsof methods in all training traces.

跟踪采样:我们的训练数据包含大量序列。因此，使用它们来构建 FSa 是非常昂贵的。因此，跟踪采样的目标是创建一个更小的子集，它可能合理地很好地代表所有跟踪的整个集合。我们提出了一种启发式方法来寻找一个轨迹子集，该子集覆盖了所有训练轨迹中的所有共现方法对。

Algorithm 1 shows our heuristic to select execution traces from the whole set of execution traces S. At first, we determine all pairs (a, b) where a and b are methods that occur together in at least one trace in S and store them in P (lines 4 to 9).Next, we create a set O that contains selected traces - initially O is an empty set (line 10).Then, we iteratively choose a pair (a, b) which does not appear in any trace in O and occur in the least number of input traces in S (line 12).Given a selected pair (a, b), we look for the shortest trace Si < O where a, b ∈ Si (line 13).Once the trace is found, we

算法 1 展示了我们从整个执行轨迹集合中选择执行轨迹的启发式方法。首先，我们确定所有对(a，b)，其中 a 和 b 是在至少一个轨迹中一起出现在 S 中的方法，并将它们存储在 P 中(第 4 行到第 9 行)。接下来，我们创建一个包含所选轨迹的集合 0——最初 0 是一个空集合(第 10 行)。然后，我们迭代地选择一对(a，b)，它不出现在 0 中的任何轨迹中，而出现在最少数量的输入轨迹中(第 12 行)。给定选定的对(a，b)，我们寻找最短的迹线 S1 < 0，其中 a，b∈S1(第 13 行)。一旦找到踪迹，我们

(m1, m2) is a co-occurrence pair if m1 and m2 appear together in at least one trace.

如果 m1 和 m2 一起出现在至少一个轨迹中，则(m1，m2)是同现对。

Feature ID Value	
Type I: Previously Invoked Methods	
Fm	1 if m is invoked before the automaton state.Otherwise, 0.
Type II: Next Methods to Invoke	
Pm	Probability computed by the learned Recur-rent Neural Network based Language Model for method m to be called after a particular au-tomaton state is reached ($0 \leq Pm \leq 1$).

特征标识值	
类型一:以前调用的方法	
调频	如果 m 在自动机状态之前被调用，则为 1。否则，0。
类型二:下一个要调用的方法	
下午	由学习的基于递归神经网络的语言模型为方法 m 计算的概率，方法 m 在达到特定的原子状态($0 \leq Pm \leq 1$)后被调用。

Feature Extraction: From method sequences of the sampled exe-cution traces, we construct a Prefix Tree Acceptor (PTA).A PTA is a tree-like deterministic finite automaton (DFA) created by putting all the prefixes of sequences as states, and a PTA only accepts the sequences that it is built from.The final states of our constructed PTAs are the ones have incoming edges with <END> labels (see Section 3.2).Figure 3 shows an example of a Prefix Tree Acceptor (PTA).Table 1 shows information of the extracted features.For each state S of a PTA, we are particularly interested in two types of features: (1) Type I: This type of features captures information of previously

特征提取:从采样的执行轨迹的方法序列中，我们构造了前缀树接受器。PTA 是一种树状确定性有限自动机，它是通过将序列的所有前缀作为状态来创建的，PTA 只接受构建它的序列。我们构建的 PTa 的最终状态是具有带<结束>标签的进入边缘的状态(见第 3.2 节)。图 3 显示了前缀树接收器(PTA)的一个例子。表 1 显示了提取特征的信息。对于 PTA 的每个状态，我们对两种类型的特征特别感兴趣:(1)类型一:这种类型的特征捕获以前的信息

invoked methods before the state S is reached.The values of type I features for state S is the occurrences of methods on the path between the starting state (i.e., the root of the PTA) and S. For example, according to Figure 3, the values of Type I features corresponding to node S3 are: F<START> = F<init> = FinitVerify = and Fupdate = Fverify = F<END> = .

在达到状态之前调用方法。状态 S 的第一类特征的值是在起始状态(即 PTA 的根)和状态 S 之间的路径上的方法的出现。例如，根据图 3，对应于节点 S3 的第一类特征的值是:F<START> = F<init> = FinitVerify =和 Fupdate = Fverify = F<END> =。

(2) Type II: This type of features captures the likely methods to be immediately called after a state is reached.Values of these features are computed by the inferred RNNLM in the

(2)第二类:这种类型的特性捕获了在达到一个状态后可能立即调用的方法。这些特征的值由

109

Algorithm 2: Predicting Precision of a finite-state automa-ton given a set of method sequence.

算法 2:给定一组方法序列，预测有限状态自动机的精度。

Input :M: an finite-state automaton

输入:M:有限状态自动机

Data: a set of training method sequences Output :Predicted Precision of M 1 PDat a ← empty set 2 for seq ∈ Data do

数据:一组训练方法序列输出:序列∈数据 do 的 m1 PDat←空集 2 的预测精度

3 for 0 ≤ i < length(seq) − 1 do 4 Include (seq[i],seq[i + 1]) to PDat a 5 end 6 end

3 对于 0 ≤ i <长度(seq)1 d0 4 包括(seq[i，seq[i + 1])到 PDat 的 5 端 6 端

7 PM ← empty set

晚上 7 点←空套

8 EM ← the set of transitions in M 9 for s1 m−−→ s2 ∈ EM ∧ s2 m−−→ s3 ∈ EM do 10 Include (m1,m2) to PM 11 end

8 EM 9 中 S1 M→S2∈EM∈S2 M→S3∈EM do 10 包括(m1，m2)到 PM 11 的一组转换结束

precision ← |PDat a | |PDat a ∪ PM | 13 return precision

精度←| PDat a | | PDat a≈PM | 13 返回精度

deep learning step (see Section 3.2).For example, at node S3 in Figure 3, initVerify and <END> have higher probabili-ties than the other methods to be called afterward.Exam-ples of type II features and their values for node S3 output by a RNNLM are as follows: PinitVerify = P<END> = .and P<START> = P<init> = Pverify = Pupdate = ..

深度学习步骤(参见第 3.2 节)。例如，在图 3 的节点 S3，initVerify 和< END >比之后调用的其他方法具有更高的概率。RNNLM 输出的节点 S3 的第二类特征及其值的例子如下:P <开始> = P <初始> = P 验证=化蛹日期=..

Our intuition of extracting different types of features is to provide sufficient information for clustering algorithms in the subsequent substep to better merge PTA nodes.

我们提取不同类型特征的直觉是为后续子步骤中的聚类算法提供足够的信息，以便更好地合并 PTA 节点。

Clustering: We run k-means [35] and hierarchical clustering [43] algorithms on the PTA's states with their extracted features.Our goal is to create a simpler and more generalized automaton that cap-tures specifications of a target library class.Since both k-means and hierarchical clustering require the predefined inputC for number of clusters, we try with many values ofC from 2 to MAX_CLUSTER (refer to Section 4.2.2) to search for the best FSA.Overall, the execution of clustering algorithms results in 2 × (MAX_CLUSTER − 1) FSAs.

聚类:我们运行 k-means [35]和分层聚类[43]算法对 PTA 的状态及其提取的特征。我们的目标是创建一个更简单、更一般化的自动机来满足目标库类的规范。由于 k 均值聚类和层次聚类都需要预定义的聚类数量输入，我们尝试使用从 2 到 MAX_CLUSTER 的许多值(参见第 4.2.2 节)来搜索最佳的 FSA。总的来说，聚类算法的执行会产生 2 ×(MAX _ CLUSTER 1)个 FSAs。

Model Selection: We propose a heuristic to select the best FSA among the ones output by the clustering algorithms.Algorithm 2 describes our strategy to predict precision of an automaton M given the set of all traces Data (see Section 3.1).

模型选择:我们提出了一种启发式算法来从聚类算法输出的结果中选择最佳的有限状态机。算法 2 描述了我们在给定所有轨迹数据集的情况下预测自动机精度的策略(参见第 3.1 节)。

We predict Precision by first constructing a set PDat a containing all pairs (m1,m2), where m1 and m2 appear consecutively (i.e., m1 is called right before m2) in an execution trace in Data.Then, we construct another set PM containing all pairs (m1,m2) that appear consecutively in a trace generated by the automaton M. In Algo-rithm 2, lines 1 to 6 compute all pairs (m1,m2) occurring in Data, and lines 7 to 11 collect those pairs occurring in traces generated by the input automaton M. To find all pairs occurring in traces gen-erated by M, we look for two transitionss1 $m{-}{-}\rightarrow$ s ′ 1 and s2 $m{-}{-}\rightarrow$ s ′ 2 of M, where s ′ 1 = s2.We take labels of the two transitions (i.e., m1 and m2) and add a pair of methods (m1,m2) to the set PM .Line 12 com-putes the predicted value of Precision, which is the ratio between of number of all pairs in PM and all pairs in PDat a ∪ PM .We input

我们通过首先构建包含所有对(m1，m2)的集 PDat a 来预测精度，其中 m1 和 m2 在数据的执行跟踪中连续出现(即 m1 正好在 m2 之前被调用)。然后，我们构造另一个包含自动机 M 生成的轨迹中连续出现的所有对(m1，m 2)的集合 PM。在算法 2 中，第 1 至 6 行计算数据中出现的所有对(m1，m2)，第 7 至 11 行收集输入自动机 M 生成的轨迹中出现的那些对。为了找到由 M 生成的轨迹中出现的所有对，我们寻找两个过渡 S1 M→S1 和 S2 M→S2，其中 S1 = S2。我们获取两个转换(即 m1 和 m2)的标签，并将一对方法(m1，m2)添加到集合 PM 中。第 12 行计算精度的预测值，精度是指在大约下午 1 点时，所有粒子对的数量与所有粒子对的数量之比。我们输入

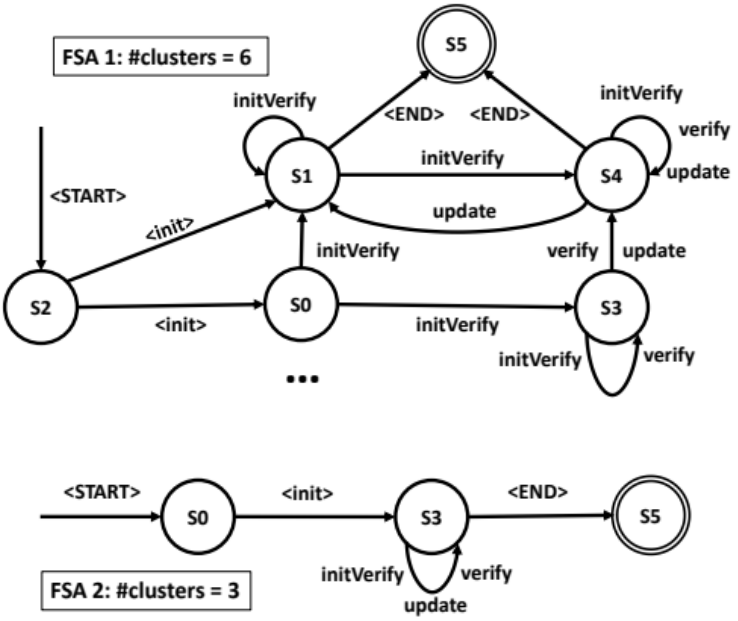


Figure 4: Example FSAs output by clustering algorithms from PTA shown in Figure 3.

图 4:通过图 3 所示的 PTA 聚类算法输出的示例 FSAs。

all FSAs created by clustering algorithms and all execution traces to Algorithm 2 for estimating the FSAs' Precision.

由聚类算法创建的所有 FSA 和算法 2 的所有执行跟踪，用于估计 FSA 的精度。

Next, we approximate the values of Recall by computing the percentage of all execution traces accepted by a given automaton M. Once all precision and recall of FSA models are predicted, we compute the expected value of F-measure (i.e., harmonic mean of precision and recall) for each of the automata.Finally, our approach returns the FSA with highest expected F-measure.

接下来，我们通过计算给定自动机 m 接受的所有执行轨迹的百分比来近似召回的值。一旦预测了 FSA 模型的所有精度和召回，我们计算每个自动机的 F 度量的期望值(即精度和召回的调和平均值)。最后，我们的方法返回具有最高期望离差的金融稳定分析。

Example: From the PTA partially shown in Figure 3, we conduct feature extraction on each state of the PTA.Then, we input these states with extracted features to clustering algorithms (i.e., k-means and hierarchical clustering) to merge similar states.If MAX_CLUSTER is identified to be 20, in total, there would be 38 FSAs constructed by the two clustering algorithms.Figure 4 shows 2 out of 19 FSAs output by k-means.Next, we perform model selection on these FSAs to select the one with highest predicted F-measures.

示例:从图 3 中部分显示的 PTA 中，我们对 PTA 的每个状态进行特征提取。然后，我们将这些具有提取特征的状态输入聚类算法(即 k 均值和分层聚类)以合并相似的状态。如果 MAX_CLUSTER 被识别为总共 20 个，那么将有 38 个由两个聚类算法构建的 FSA。图 4 显示了 k 均值输出的 19 个 FSa 中的 2 个。接下来，我们对这些金融服务协议进行模型选择，以选择具有最高预测金融服务措施的金融服务协议。

4 EMPIRICAL EVALUATION

4 实证评估

4.1 Dataset

4.1 数据集

4.1.1 Target Library Classes.In our experiments, we select 11 target library classes as the benchmark to evaluate the effectiveness of our proposed approach.These library classes were investigated by previous research works in specification mining [24, 26].Table 2 shows further details of the selected library classes including infor-mation of collected execution traces.Among these library classes, 9 out of 11 are from Java Development Kit (JDK);the other two library classes are DataStructure.StackAr (from Daikon project) and NumberFormatStringTokenizer (from Apache Xalan).For every library class, we consider methods that were analyzed by Krka et al. [24].

4.1.1 目标库类。在我们的实验中，我们选择了 11 个目标库类作为基准来评估我们提出的方法的有效性。这些库类由先前在规范挖掘[24，26]中的研究工作调查。表 2 显示了所选库类的更多细节，包括收集的执行跟踪信息。在这些库类中，11 个中有 9 个来自 Java 开发工具包(JDK)；另外两个库类是 DataStructure。StackAr(来自 Daikon 项目)和 NumberFormatStringTokenizer(来自 Apache Xalan)。对于每一个图书馆类，我们都考虑由克尔卡等人[24]分析的方法。

4.1.2 Ground Truth Models.We utilize ground truth mod-els created by Krka et al. [24].Among the investigated library classes, we refine ground truth models of five Java's Collec-tion based library classes (i.e., ArrayList, LinkedList, HashMap, HashSet, and Hashtable) to capture "empty" and "non-empty"

4.1.2 基本事实模型。我们利用克尔卡等人[24]创建的地面真相模型。在调查的库类中，我们细化了五个基于 Java 集合的库类(即数组列表、链接列表、哈希表、哈希表和哈希表)的基本事实模型，以捕获“空”和“非空”

110

Table 2: Target Library Classes."#M" represents the num-ber of class methods that are analyzed, "#Generated Test Cases" is the number of test cases generated by Ran-doop, "#Recorded Method Calls" is the number of recorded method calls in the execution traces, "NFST" stands for NumberFormatStringTokenizer.

表 2:目标库类。“#M”代表被分析的类方法的数量，“生成的测试用例”是由 Ran-doop 生成的测试用例的数量，“记录的方法调用”是执行跟踪中记录的方法调用的数量，“NFST”代表 NumberFormatStringTokenizer。

Target Library #M #Generated #Recorded Class Test Cases Method Calls ArrayList 18 42,865 22,996 HashMap 11 53,396 67,942 Hashtable 8 79,403 89,811 HashSet 8 23,181 257,428 LinkedList 7 13,731 4,847 NFST 5 15,8998 95,149 Signature 5 79,096 205,386 Socket 21 80,035 130,876 StringTokenizer 5 148,649 336,924 StackAr 7 549,648 13,2826 ZipOutputStream 5 162,971 43,626

目标库# M #生成#记录类测试用例方法调用数组列表 18 42，865 22，996 哈希表 11 53，396 67，942 哈希表 8 79，403 89，811 哈希表 8 23，181 257，428 链接列表 7 13，731 4，847 NFST 5 15，8998 95，149 签名 5 79，096 205，386 套接字

states of Collection objects.We also revise ground truth mod-els of NumberFormatStringTokenizer and Socket by including missing transitions of the original models.

集合对象的状态。我们还修改了 NumberFormatStringTokenizer 和 Socket 的基本事实模型，包括原始模型的缺失转换。

4.2 Experimental Settings

4.2 实验设置

4.2.1 Evaluation Metrics.We follow Lo and Khoo's method [32] to measure precision and recall for assessing the effectiveness of our proposed approach.Lo and Khoo's method has been widely adopted by many prior specification mining works [24, 27].Their proposed approach takes as input a ground truth and an inferred FSA.Next, it generates sentences (i.e., traces) from the two FSAs to compute their similarities.Precision of an inferred FSA is the percentage of sentences accepted by its corresponding ground truth model among the ones that are generated by that FSA.Recall of an inferred FSA is the percentage of sentences accepted by itself among the ones that are generated by the corresponding ground truth model.In a nutshell, precision reflects the percentage of sen-tences produced by an inferred model that are correct, while recall reflects the percentage of correct sentences that an inferred model can produce.We use F-measure, which is the harmonic mean of precision and recall, as a summary metric to evaluate specification mining algorithms.F-measure is defined as follows:

4.2.1 评估指标。我们遵循劳和霍的方法[32]来测量精确度和召回率，以评估我们建议的方法的有效性。洛和霍的方法已被[24，27]的许多先前规范采矿工程广泛采用。他们提出的方法将一个基本事实和一个推断的金融稳定分析作为输入。接下来，它从两个 FSa 生成句子(即轨迹)，以计算它们的相似性。推断的模糊推理的精确度是其相应的基本事实模型所接受的句子在该模糊推理生成的句子中所占的百分比。回忆一个推断的 FSA 是在由相应的基本事实模型产生的句子中，它自己接受的句子的百分比。简而言之，精确度反映了推断模型产生的正确感觉的百分比，而回忆反映了推断模型能够产生的正确句子的百分比。我们使用精确度和召回率的调和平均值——F-measure 作为评估规范挖掘算法的总结度量。f-测量定义如下:

F-Measure = 2 ×

f-测量= 2 倍

Precision × Recall

精确度×召回率

Precision + Recall (1)

精确+召回(1)

To accurately compute precision, recall and F-measure, sentences generated from a FSA must thoroughly cover its states and transi-tions.To achieve that goal, we set the maximum number of gener-ated sentences to 10,000 with maximal length of 50, and minimum coverage of each transition equals to 20.Similar strategies were adopted in prior works [24, 26].

为了精确地计算精确度、回忆和离差，从一个句子生成的句子必须完全覆盖它的状态和转换。为了实现这个目标，我们将最大生成句子数设置为 10，000，最大长度为 50，每个转换的最小覆盖范围等于 20。[24，26]以前的著作也采用了类似的策略。

4.2.2 Experimental Configurations & Environments.

4.2.2 实验配置和环境。

Randoop Configuration.In test case generation step, for each target class, we repeatedly execute Randoop (version 3.1.2) with a time limit of 5 minute with 20 different initial seeds.We set the

随机配置。在测试用例生成步骤中，对于每个目标类，我们用 20 个不同的初始种子重复执行 Randoop(3 . 1 . 2 版)，时间限制为 5 分钟。我们设置了

time limit to 5 minutes to make sure subsequent collected execution traces are not too long as well as not too short.We repeat execu-tion of Randoop 20 times to maximize the coverage of possible sequences of program methods in Randoop generated test cases.Furthermore, we turn off Randoop's option of generating error-revealing test cases (i.e., - no-error-revealing-tests is set to true) as executions of these test cases are usually interrupted by exceptions or errors, which results in incomplete method sequences for subsequent deep learning process.We find that with this setup the generated traces cover 100% of target API methods;also, on average, 96.97% and 98.18% of edges and states in each target class' ground-truth model are covered.

时间限制为 5 分钟，以确保后续收集的执行跟踪不太长也不太短。我们重复执行 Randoop 20 次，以最大限度地覆盖 Randoop 生成的测试用例中可能的程序方法序列。此外，我们关闭了 Randoop 生成错误揭示测试用例的选项(即，无错误揭示测试设置为真)，因为这些测试用例的执行通常被异常或错误中断，这导致后续深度学习过程的方法序列不完整。我们发现在这个设置中，生成的跟踪覆盖了 100%的目标应用编程接口方法；此外，平均而言，每个目标类的基本真理模型中 96.97%和 98.18%的边和状态都被覆盖。

RNNLM.We use a publicly available implementation of RNNLM based on TensorFlow.We execute this implementation on a NVIDIA DGX-1 Deep Learning system.We use the default con-figuration included as part of the implementation.

RNNLM。我们使用基于张量流的 RNNLM 的公开实现。我们在英伟达 DGX-1 深度学习系统上执行这一实施。我们使用作为实现一部分的默认配置。

Clustering Configuration.In clustering step, we run k-means and hierarchical clustering with the setting of number of clusters from 2 to MAX_CLUSTER = 20 for ev-ery target class.We use sklearn.cluster.KMeans and sklearn.cluster.AgglomerativeClustering of scikit-learn (version 0.18) with default settings.

群集配置。在聚类步骤中，我们运行 k-means 和分层聚类，每个目标类的聚类数设置为从 2 到 MAX_CLUSTER = 20。我们使用带有默认设置的 sci kit-learn(0.18 版)的 sklearn.cluster.KMeans 和 sklearn . cluster . aggregation 群集。

4.2.3 Baselines.In the experiments, we compare the effective-ness of DSM with many previous specification mining works .Krka et al. propose a number of algorithms that analyze execu-tion traces to infer FSAs [24].These algorithms are k-tails, CON-TRACTOR++, SEKT, and TEMI.CONTRACTOR++, TEMI, and SEKT infer models leveraging invariants learned using Daikon.On the other hand, k-tails construct models only from ordering of methods in execution traces.Despite the fact that DSM is not processing likely invariants, we include CONTRACTOR++,

SEKT, and TEMI as baselines to compare the applicability of deep learning and likely invariant inference in specification mining.For k-tails and SEKT, we choose k ∈ {1, 2} following Krka et al. [24] and Le et al. [26]'s configurations.In total, we have six different baselines: Traditional 1-tails, Traditional 2-tails, CONTRACTOR++, SEKT 1-tails.SEKT 2-tails, and TEMI

4.2.3 基线。在实验中，我们将需求侧管理的有效性与许多以前的规范挖掘工作进行了比较。克尔卡等人提出了许多分析执行轨迹的算法来推断 FSAs [24]。这些算法是 k-tails、CON-TRACTOR++、SEKT 和 TEMI。承包商++、TEMI 和塞科特利用戴康学到的不变量推断模型。另一方面，k-tails 仅根据执行跟踪中方法的顺序构建模型。尽管需求侧管理没有处理可能的不变量，但我们将 CONTRACTOR++、SEKT 和 TEMI 作为基线，比较深度学习和可能的不变量推理在规范挖掘中的适用性。对于 k-tails 和 SEKT，我们选择 k∑{ 1，2}跟随 Krka 等人的[24]和 Le 等人的[26 的配置。总的来说，我们有六种不同的基线:传统的 1 尾、传统的 2 尾、CONTRACTOR++、SEKT 1 尾。SEKT 2-tails 和 TEMI

We use the implementation of provided by Krka et al.[24].We utilize Daikon [14] to collect execution traces from Randoop generated test cases and inferred likely invariants from all of the traces.Originally, Krka et al.'s implementation uses a 32-bit version of Yices 1.0 Java Lite API, which only works with 32-bit Java Virtual Machine and limited to use up to 4 GB heap memory.Since the amount of execution traces is large, we follow two experimental schemes to run Krka et al.'s code:

我们使用 Krka 等人提供的实现[[24]。我们利用 Daikon [14]从 Randoop 生成的测试用例中收集执行跟踪，并从所有跟踪中推断出可能的不变量。最初，Krka 等人的实现使用 32 位版本的 ices 1.0 Java Lite 应用编程接口，该接口仅适用于 32 位的 Java 虚拟机，并且最多只能使用 4 GB 的堆内存。由于执行跟踪的数量很大，我们遵循两个实验方案来运行 Krka 等人的代码:

(1) Scheme I: Krka et al.'s implementation is updated to work with the 64-bit libraries of Yices 1 SMT solver.Then, we input execution traces of all generated test case as well as Daikon invariants inferred by these traces to all baselines.For each application of Krka et al.'s code, we set the maximum allocated memory to 7 GB and time limit to 12 hours.

(1)方案一:Krka 等人的实现被更新以与 ices 1 SMT 求解器的 64 位库一起工作。然后，我们将所有生成的测试用例的执行轨迹以及由这些轨迹推断的 Daikon 不变量输入到所有基线。对于 Krka 等人代码的每个应用，我们将最大分配内存设置为 7 GB，时间限制为 12 小时。

https://github.com/hunkim/word-rnn-tensorflow http://softarch.usc.edu/wiki/doku.php?id=inference:start http://atlantis.seidenberg.pace.edu/wiki/lep/Yices_Java_API_Lite http://yices.csl.sri.com/old/download-yices1.shtml

https://github.com/hunkim/word-rnn-tensorflow·http://softarch.usc.edu/wiki/doku.php?id=inference:start

111

111

Table 3: Effectiveness of DSM."F" is F-measure.Class F (%) Class F (%) ArrayList 22.21 Signature 100.00 HashMap 86.71 Socket 54.24 HashSet 76.84 StackAr 74.38 Hashtable 79.92 StringTokenizer 100.00 LinkedList 30.98 ZipOutputStream 88.82 NFST 77.52% Average 71.97

表 3:需求侧管理的有效性。“福”是福的量度。F 类(%)F 类(%)数组列表 22.21 签名 100.00 哈希表 86.71 套接字 54.24 哈希表 76.84 堆栈卡 74.38 哈希表 79.92 字符串化器 100.00 链接列表 30.98 输出流 88.82 NFST 77.52%平均 71.97

(2) Scheme II: We use the original Krka et al.'s implementation and a set of execution traces corresponding to test cases generated by Randoop with one specific seed.

(2)方案二:我们使用原始的 Krka 等人的实现和一组对应于兰多普用一个特定种子生成的测试用例的执行轨迹。

4.3 Research Questions

4.3 研究问题

RQ1: How effective is DSM?In this research question, we com-pute precision, recall, and F-measure of FSAs inferred by our ap-proach for the 11 target library classes.

问题 1:需求侧管理有多有效？在这个研究问题中，我们计算了用我们的方法为 11 个目标库类推断的 FSA 的精度、召回率和 f 测度。

RQ2: How does DSM compare to existing specification min-ing algorithms?In this research question, we compare DSM with a number of existing specification mining algorithms proposed by Krka et al. [24] in various experimental schemes.RQ3: Which Recurrent Neural Network (RNN) is best for DSM?By default, DSM trains RNNLMs using Long-Short Term Memory (LSTM) networks from execution traces.In this research question, we first adapt DSM to use the standard RNN and Gated Recurrent Units (GRU) [9] networks for constructing language mod-els with the same learning configuration as LSTM (see Section 3.2).Then, we analyze the effectiveness of DSM for each neural network architecture (i.e., Standard RNN, LSTM, and GRU).

RQ2:需求侧管理与现有的规格最小化算法相比如何？在这个研究问题中，我们在各种实验方案中将需求侧管理与 Krka 等人提出的许多现有规范挖掘算法进行了比较。RQ3:哪种递归神经网络(RNN)最适合需求侧管理？默认情况下，DSM 使用来自执行跟踪的长短期内存(LSTM)网络来训练 RNNLMs。在这个研究问题中，我们首先调整需求侧管理以使用标准的 RNN 和门控循环单元(GRU) [9]网络来构建具有与 LSTM 相同学习配置的语言模型(见第 3.2 节)。然后，我们分析了需求侧管理对于每个神经网络体系结构(即标准 RNN、LSTM 和 GRU)的有效性。

4.4 Findings

4.4 调查结果

RQ1: DSM's Effectiveness.Table 3 shows the F-measure of DSM for the eleven target library classes (Section 4.1).From the table, our approach achieves an average F-measure of 71.97%.Noticeably, for StringTokenizer and Signature, DSM infers models that exactly match ground truth models (i.e., F-measure of 100%).There are other 6 out of the 11 library classes where our approach achieves F-measure of 70% or greater.

RQ1:需求侧管理的有效性。表 3 显示了 11 个目标库类的需求侧管理的 F 值(第 4.1 节)。从表中可以看出，我们的方法达到了 71.97%的平均离差。值得注意的是，对于字符串化和签名，需求侧管理推断出与基本事实模型完全匹配的模型(即 100%的离差度量)。在 11 个库类中，还有 6 个我们的方法达到了 70%或更高的 F 值。

By taking a closer look into DSM intermediary results, we find that across the 11 classes, DSM performs 26 - 1,072 merge operations (with an average of 303.45 operations) to construct the final FSAs from the PTAs.Also, the differences in the predicted F-measures of the FSAs produced in the clustering step range between 22.05% - 68.32% (with an average of 39.13%).These shows that DSM com-ponents need to perform substantial amount of work to bring the PTAs to the final FSAs.

通过仔细研究需求侧管理的中间结果，我们发现在 11 个类中，需求侧管理执行 26 - 1，072 个合并操作(平均 303.45 个操作)来从 PTa 构建最终的 FSa。此外，在聚类步骤中产生的金融服务协议的预测金融服务措施的差异在 22.05% - 68.32%之间(平均为 39.13%)。这些表明，需求侧管理部门需要做大量的工作才能使家长教师协会达到最终的家庭服务协议。

RQ2: DSM vs. Previous Works.

RQ2:需求侧管理与前期工作。

Scheme I: The first part of Table 4 highlights the F-measure of 6 baselines proposed by Krka et al. [24] following Scheme I. In this experimental scheme, we use all of execution traces that we collect from generated test cases and likely invariants inferred from these traces as input data to the baselines.According to the figure, CONTRACTOR++ is the most effective baseline in terms of average F-measure (i.e., 55.22%) in this experimental scheme;Traditional 1-tails is the best performing baseline that only uses execution traces to construct FSAs (i.e., average F-measure of 33.42%).DSM

方案一:表 4 的第一部分强调了 Krka 等人根据方案一提出的 6 个基线的 F 度量。在这个实验方案中，我们使用从生成的测试用例中收集的所有执行轨迹和从这些轨迹推断的可能不变量作为基线的输入数据。根据该图，承包商++是该实验方案中平均测得的最有效基线(即 55.22%)；传统的单尾是性能最好的基线，它只使用执行轨迹来构建 FSA(即平均 F 度量为 33.42%)。设计标准手册

is more effective than all of the baselines in terms of average pre-cision, recall, and F-measure.In terms of average F-measure, we outperform CONTRACTOR++ (i.e., the best performing baseline) and Traditional 1-tails (i.e., the best baseline that only use execution traces) by 30.33% and 115.35%, respectively.Furthermore, we note many baselines have no available results for precision, recall, and F-measure.This is because these baselines are not able to return FSAs mostly due to out of heap memory errors or time limit exceeded errors.

在平均预测、召回和离差度量方面比所有基线都更有效。就平均离差度量而言，我们的表现分别优于承包商++(即最佳绩效基线)和传统单尾(即仅使用执行轨迹的最佳基线)30.33%和 115.35%。此外，我们注意到许多基线没有精度、召回率和离差测量的可用结果。这是因为这些基线无法返回 FSa，主要是因为堆内存不足或时间限制超出了错误。

Scheme II.The second part of Table 4 shows the F-measure of 6 baselines proposed by Krka et al. [24] following Scheme II.Opti-mistic TEMI is the best performing baseline in terms of average F-measure;Traditional 1-tails is the best baseline that constructs models from execution traces with the average F-measure of 53.97%.In comparison with DSM, we note that the average recall, and F-measure of our approach is higher than those of all the six baselines.That indicates our inferred models are more generalized and less overfitted.In terms of average F-measure, our approach outper-forms the best baseline (i.e., Optimistic TEMI) by 28.22%.DSM is also more effective than Traditional 1-tails (i.e., the best baseline that infers FSAs only from method orderings in traces) by 31.57%.Noticeably, compared to the baselines that construct automata from execution traces (i.e., Traditional 1-tails and Traditional 2-tails), DSM's F-measures are higher for all target library classes.For the other baselines that use likely invariants, our approach is more effective in terms of F-measures for 7 out of 11 target library classes except ArrayList, LinkedList, Hashtable, and Socket.RQ3: Best RNN Architecture.Table 5 shows the effectiveness of DSM configured with the

three different RNN architectures.We can note that DSMLSTM (our default configuration) and DSMGRU outperform DSMRNN in terms of F-measure by 7.92% and 6.88% respectively.DSMGRU performs almost equally as well as DSMLSTM.

方案二。表 4 的第二部分显示了克尔卡等人[[24]根据方案二提出的 6 个基线的氟测量。从平均离差来看，最佳 TEMI 是表现最好的基线；传统的单尾是从执行轨迹构建模型的最佳基线，平均离差为 53.97%。与需求侧管理相比，我们注意到我们方法的平均召回率和离差度量高于所有六个基线。这表明我们推断的模型更一般化，也更不过度。就平均离差度量而言，我们的方法比最佳基线(即乐观 TEMI)高出 28.22%。需求侧管理也比传统的单尾(即仅从跟踪中的方法顺序推断 FSa 的最佳基线)更有效 31.57%。值得注意的是，与从执行轨迹构造自动机的基线(即传统的 1 尾和传统的 2 尾)相比，DSM 对所有目标库类的 F 度量都更高。对于使用可能不变量的其他基线，我们的方法对于除数组列表、链接列表、哈希表和套接字之外的 11 个目标库类中的 7 个更有效。RQ3:最佳 RNN 建筑。表 5 显示了用三种不同的 RNN 架构配置的需求侧管理的有效性。我们可以注意到，DSMLSTM(我们的默认配置)和 DSMGRU 在 F 度量方面的表现分别比 DSMRNN 好 7.92%和 6.88%。帝斯曼集团的表现几乎和帝斯曼一样。

5 MINING FSA FOR DETECTING ANDROID MALICIOUS BEHAVIORS

5 用于检测 ANDROID 恶意行为的挖掘 FSA

Nowadays, Android is the most popular mobile platform with mil-lions of apps and supported devices.As the matter of fact, An-droid users easily become targets of attackers.Recently, several approaches have been proposed to protect users from potential threats of malware.Among state-of-the-art approaches, Jamrozik et al. propose Boxmate that mines rules to construct Android sand-boxes by exploring behaviors of target benign apps [21].The key idea of Boxmate is it prevents a program to change its behaviour;it can prevent hidden attacks, backdoors, and exploited vulnerabili-ties from compromising the security of an Android app.Boxmate works on two phases: monitoring and deployment.In the moni-toring phase, Boxmate employs a test case generation tool, named Droidmate [22], to create a rich set of GUI test cases.During the ex-ecution of these test cases, Boxmate records invocations of sensitive API methods (e.g., methods that access cameras, locations, etc.), and use them to create sandbox rules.The rules specify what sensitive API methods are allowed to be invoked during deployment.During deployment, when an app accesses a sensitive API method that is not recorded in the above rules, the sandbox immediately inter-cepts that operation and raises warning messages to users about the suspicious activity.

如今，安卓是最受欢迎的移动平台，拥有数百万个应用和支持的设备。事实上，机器人用户很容易成为攻击者的目标。最近，已经提出了几种方法来保护用户免受恶意软件的潜在威胁。在最先进的方法中，Jamrozik 等人提出 Boxmate，通过探索目标良性应用的行为来挖掘构建安卓沙盒的规则，[21]。Boxmate 的关键思想是它阻止一个程序改变它的行为；它可以防止隐藏的攻击、后门和利用漏洞危及安卓应用的安全性。Boxmate 分两个阶段工作:监控和部署。在监控阶段，Boxmate 使用一个名为 Droidmate [22 的测试用例生成工具来创建一组丰富的图形用户界面测试用例。在这些测试用例的执行过程中，Boxmate 记录敏感的应用编程接口方法的调用(例如，访问摄像机、位置等的方法)。)，并使用它们创建沙箱规则。这些规则指定了在部署期间允许调用哪些敏感的应用编程接口方法。在部署期间，当应用程序访问上述规则中未记录的敏感应用编程接口方法时，沙箱会立即接受该操作，并向用户发出关于可疑活动的警告消息。

112

ISSTA'18，2018 年 7 月 16-21 日，阿姆斯特丹，荷兰蒂恩杜伊勒和卢大伟

Table 4: F-measure (%): Scheme I vs. Scheme II."T1" is Traditional 1-tails, "T2" is Traditional 2-tails, "C+" is CONTRACTOR++, "S1" is SEKT 1-tails, "S2" is SEKT 2-tails, "OT" is Optimistic TEMI, "NFST" is NumberFormatStringTokenizer, "-" means that the result is not available.

表 4:离差(%):方案一与方案二。“T1”是传统的 1-tails，“T2”是传统的 2-tails，“C+”是 CONTRACTOR++，“S1”是 SEKT 1-tails，“S2”是 SEKT 2-tails，“OT”是乐观的 TEMI，“NFST”是数字格式字符串化者，“-”表示结果不可用。

Target Scheme I Scheme II

目标方案一方案二

Library Class T1 T2 C+ S1 S2 OT T1 T2 C+ S1 S2 OT

图书馆等级 T1 T2·C+S1·S2·OT T1 T2·C+S1·S2·OT

ArrayList 13.96 13.13 36.03 13.86 13.07 16.87 5.61 3.08 36.03 4.34 3.08 35.82

数组列表 13.96 13.13 36.03 13.86 13.07 16.87 5.61 3.08 36.03 4.34 3.08 35.82

HashMap 25.41 8.71 68.94 37.35 21.93 68.94 37.35 21.93 68.94

哈希表 25.41 8.71 68.94 37.35 21.93 68.94 37.35 21.93 68.94

HashSet 20.88 21.27 52.22 20.88 21.27 23.34 22.96 15.35 52.22 22.96 15.35 55.62

HashSet 20.88 21.27 52.22 20.88 21.27 23.34 22.96 15.35 52.22 22.96 15.35 55.62

Hashtable 42.39 33.58 92.78 78.42 73.37 92.78 81.69 65.47 92.78

哈希表 42.39 33.58 92.78 78.42 73.37 92.78 81.69 65.47 92.78

LinkedList 27.15 25.72 86.02 26.67 24.52 7.51 26.03 8.07 86.02 18.45 6.59 86.02

链接列表 27.15 25.72 86.02 26.67 24.52 7.51 26.03 8.07 86.02 18.45 6.59 86.02

NFST 24.57 25.52 30.40 24.56 25.78 11.80 68.72 51.04 30.40 66.58 49.51 33.40

NFST 24.57 25.52 30.40 24.56 25.78 11.80 68.72 51.04 30.40 66.58 49.51 33.40

Signature 61.54 64.25 66.88 62.05 63.98 39.06 100 95.41 66.88 95.41 89.78 66.88

签名 61.54 64.25 66.88 62.05 63.98 39.06 100 95.41 66.88 95.41 89.78 66.88

Socket 35.89 31.52 55.15 34.73 28.37 37.30 21.98 55.15 35.32 20.87 55.62

插座 35.89 31.52 55.15 34.73 28.37 37.30 21.98 55.15 35.32 20.87 55.62

StackAr 16.54 16.54 34.91 16.54 16.54 42.57 42.57 34.91 42.57 42.57

stack ar 16.54 16.54 34.91 16.54 16.54 42.57 42.57 34.91 42.57 42.57

StringTokenizer 52.88 52.97 21.30 52.15 100 89.69 21.30 92.21 84.77 0.00

stringTokenizer 52.88 52.97 21.30 52.15 100 89.69 21.30 92.21 84.77 0.00

ZipOutputStream 46.36 47.42 62.80 47.91 82.51 78.08 62.08 86.47 67.84 66.20

zip outputStream 46.36 47.42 62.80 47.91 82.51 78.08 62.08 86.47 67.84 66.20

Average 33.42 30.97 55.22 33.26 27.65 19.72 54.70 45.50 55.22 53.03 42.52 56.13

平均 33.42 30.97 55.22 33.26 27.65 19.72 54.70 45.50 55.22 53.03 42.52 56.13

Table 5: F-measure (%): DSM with standard RNN (DSMRNN), LSTM (DSMLSTM), and GRU (DSMGRU).

表 5:离差(%):标准 RNN、LSTM 和 GRU 的需求侧管理。

Target Library DSMLSTM DSMRNN DSMGRU Class ArrayList 22.21 4.95 9.39 HashMap 86.71 97.76 100.00 HashSet 76.84 68.86 73.88 Hashtable 79.92 87.94 87.94 LinkedList 30.98 32.29 34.86 NFST 77.52 70.09 73.31 Signature 100.00 65.31 95.41 Socket 54.24 51.31 58.34 StackAr 74.38 71.79 77.67 StringTokenizer 100.00 95.88 100.00 ZipOutputStream 88.82 87.36 73.29 Average 71.97 66.69 71.28

目标库 DSMLSTM DSMRNN DSMGRU 类数组列表 22.21 4.95 9.39 哈希表 86.71 97.76 100.00 哈希表 76.84 68.86 73.88 哈希表 79.92 87.94 87.94 链接列表 30.98 32.29 34.86 NFST 77.52 70.09 73.31 签名 100.00 65

Boxmate's mined sandbox rules constitute a behavior model that accepts a sensitive API method as input and predicts its invocation as benign or malicious.Nevertheless, we find that attackers can bypass Boxmate's models if they can perform malicious activities by invoking the same sensitive APIs as those used by the apps during their normal operations.Therefore, we propose a technique to employ FSAs inferred by DSM to construct more comprehensive sandboxes that consider the context of sensitive API invocations namely series of methods called prior to the sensitive API invoca-tions - that can better protect Android users from attackers.For each sensitive API method M, we select W previously executed non-sensitive API methods before M to create training traces.Then, we group all of the training traces into one single set and input it to DSM.The output of DSM is a FSA that captures interactions between sensitive and non-sensitive API methods.The FSA is then leveraged as an automaton based behavior model to build a sandbox for more effective protection.

Boxmate 挖掘的沙箱规则构成了一个行为模型，它接受敏感的应用编程接口方法作为输入，并预测其调用是良性的还是恶意的。然而，我们发现攻击者可以绕过 Boxmate 的模型，如果他们可以通过调用应用程序在正常操作中使用的敏感 APIs 来执行恶意活动的话。因此，我们提出了一种技术来使用 DSM 推断的 FSAs 来构建更全面的沙箱，该沙箱考虑敏感的应用编程接口调用的上下文，即在敏感的应用编程接口调用之前调用的一系列方法，可以更好地保护安卓用户免受攻击者的攻击。对于每个敏感的应用编程接口方法，我们在创建训练跟踪之前选择先前执行的非敏感的应用编程接口方法。然后，我们将所有的训练轨迹分组到一个单独的集合中，并将其输入到需求侧管理。需求侧管理的输出是一个捕捉敏感和非敏感应用编程接口方法之间相互作用的前端服务协议。然后，FSA 被用作基于自动机的行为模型，以构建沙箱来实现更有效的保护。

By default, we set W = 3.

默认情况下，我们设置 W = 3。

APPS Behavioral Model Inference

应用行为模型推断

Benign Apps

良性应用

DSM

设计标准手册

Monitoring Phase Deployment Phase

监控阶段部署阶段

Execution traces of sensitive & relevant APIs

敏感和相关 APIs 的执行跟踪

Finite State Automata

有限状态自动机

APPS

APPS

Apps

应用程序

Suspicious Malicious Behaviors

可疑的恶意行为

S A N D B O X

美国药品管理局

Execution traces of sensitive & relevant APIs

敏感和相关 APIs 的执行跟踪

Rejected

拒绝

LOG

原木

LOG

原木

Figure 5: Malware detection framework leveraging behavior models inferred by DSM

图 5:利用需求侧管理推断的行为模型的恶意软件检测框架

Figure 5 demonstrates the proposed framework of our malware detection system.According to the figure, our framework has two phases:

图 5 展示了我们恶意软件检测系统的框架。根据该图，我们的框架有两个阶段:

• Monitoring Phase: This phase accepts as input a benign ver-sion of the target Android app.We first leverage GUI test case generation tools (i.e., Monkey [1], GUIRipper [2], PUMA [18], Droidmate [22], and Droidbot [30]) to create a diverse set of test cases.Next, we execute the input app with generated test cases and monitor API methods called.In particular, every time the app invokes a sensitive API method X, we select a sequence ofW previously executed API methods before X and include them to the training traces.Then, we employ DSM's mining algorithm on the gathered traces to construct a FSA based behavior model BM.The constructed model reflects behaviors of the app when calling sensitive API methods.Subsequently, we employ BM to guide an

监控阶段:该阶段接受目标安卓应用程序的良性版本作为输入。我们首先利用图形用户界面测试用例生成工具(即猴子[1 号、开膛手[2 号、彪马[18 号、[机器人 22 号和[机器人 30 号)来创建一组多样化的测试用例。接下来，我们用生成的测试用例执行输入应用程序，并监控调用的 API 方法。特别是，每次应用程序调用一个敏感的应用编程接口方法时，我们都会选择一系列之前执行的应用编程接口方法，并将它们包含到训练跟踪中。然后，利用需求侧管理对收集到的轨迹的挖掘算法，构建了一个基于有限状态机的行为模型。构建的模型反映了调用敏感的应用程序接口方法时应用程序的行为。随后，我们使用 BM 来指导

113

automaton based sandbox in the deployment phase for malware detection.

恶意软件检测部署阶段基于自动机的沙箱。

• Deployment Phase: In this phase, our framework leverages the inferred model BM to build an automaton based sandbox.The sandbox is used to govern and control execution of an Android app.Every time an app invokes a sensitive API X, the sandbox selects the sequence of W previously executed methods before X, and input them to the behavior model BM to classify the invocation of X as malicious or benign.If execution of X is pre-dicted as malicious by model BM, the sandbox

informs users by raising warning messages about suspicious activities.Otherwise, the sandbox allows the app to continue its executions without notifying users.

部署阶段:在这个阶段，我们的框架利用推断模型 BM 来构建基于自动机的沙箱。沙箱用于管理和控制安卓应用的执行。每当一个应用程序调用一个敏感的 API X 时，沙箱会在 X 之前选择一系列先前执行的方法，并将它们输入到行为模型 BM 中，将对 X 的调用分为恶意调用和良性调用。如果模型 BM 预先断定 X 的执行是恶意的，沙箱通过发出关于可疑活动的警告消息来通知用户。否则，沙箱允许应用程序在不通知用户的情况下继续执行。

We evaluate our proposed malware detection framework using a dataset of 102 pairs of Android apps that were originally collected by Li et al. [29].Each pair of apps contains one benign app and its corresponding malicious version.The malicious apps are cre-ated by injecting malicious code to their corresponding unpacked benign apps [29].All these apps are real apps that are released to various app markets.Recently, Bao et al. [4] used the above 102 pairs to assess the effectiveness of Boxmate's mined rules with 5 different test case generation tools (i.e., Monkey [1], GUIRipper [2], PUMA [18], Droidmate [22], and Droidbot [30]).In our evaluation, we utilize execution traces of the 102 Android app pairs collected by Bao et al. [4] .We setW (i.e., number of selected methods before an invoked sensitive API method) to 3, and employ these traces to infer several behavior models by using Boxmate, DSM as well as k-tails (k = 1).We include k-tails (k = 1) since according to Table 4 this is the best baseline mining algorithm that infers FSAs from raw traces of API invocations.We let the comparison between DSM and invariant based miners (i.e., CONTRACTOR++ and TEMI) for future work as Daikon is currently unable to mine invariants for Android apps.Next, we evaluate the effectiveness of inferred be-havior models in detecting malware using the following evaluation metrics:

我们使用最初由李等人[收集的 102 对安卓应用的数据集来评估我们提出的恶意软件检测框架。每对应用程序包含一个良性应用程序及其相应的恶意版本。恶意应用程序是通过将恶意代码注入相应的未打包良性应用程序而产生的[29]。所有这些应用都是发布到不同应用市场的真实应用。最近，鲍等人[4]利用上述 102 对，用 5 种不同的测试用例生成工具(即猴子[1、开膛手[2、美洲狮[18、机器人[22 和机器人[30])来评估 Boxmate 挖掘规则的有效性。在我们的评估中，我们利用了包等人[4]收集的 102 个安卓应用对的执行轨迹。我们将 w(即在调用敏感的应用编程接口方法之前选择的方法的数量)设置为 3，并使用 Boxmate、DSM 以及 k-tails (k = 1)来使用这些轨迹来推断几个行为模型。我们包括 k-tails (k = 1)，因为根据表 4，这是从 API 调用的原始跟踪中推断 FSA 的最佳基线挖掘算法。我们让需求侧管理和基于不变量的挖掘器(即承包商++和 TEMI)之间的比较用于未来的工作，因为 Daikon 目前无法为安卓应用挖掘不变量。接下来，我们使用以下评估指标评估推断行为模型在检测恶意软件方面的有效性:

• True Positive Rate (TPR): Percentage of apps that are correctly classified as malicious.TPR is computed as follows T PR =

真实阳性率(TPR):被正确归类为恶意应用的百分比。总生产率计算如下

T P

T P

T P + FN (2)

T P + FN (2)

• False Positive Rate (FPR): Percentage of apps that are incor-rectly classified as malicious.FPR is computed as follows

假阳性率(FPR):被直接归类为恶意的应用的百分比。FPR 的计算方法如下

FPR =

FPR =

FP

FP

FP +T N (3)

FP +T N (3)

In the above equations, TP (true positives) refers to the number of malicious apps that are classified as malicious, TN (true negatives) refers to the number of benign apps that are classified as benign, FP (false positives) refers to the number of benign apps are classi-fied as malicious, and FN (false negatives) refers to the number of malicious apps that are classified as benign.Both TPR and FPR are important;TPR is important as otherwise malicious behaviors are permitted, while FPR is as important as otherwise users would be annoyed with false warnings.The two metrics are well-known and widely adopted by state-of-the-art approaches in Android malware detection (e.g., [3, 50]).

在上式中，TP(真阳性)指被归类为恶意的恶意应用的数量，TN(真阴性)指被归类为良性的良性应用的数量，FP(假阳性)指被归类为恶意的良性应用的数量，FN(假阴性)指被归类为良性的恶意应用的数量。主题方案审查和 FPR 都很重要；TPR 是重要的，因为否则恶意行为是允许的，而 FPR 是重要的，因为否则用户会对错误的警告感到恼火。这两个指标是众所周知的，并且被安卓恶意软件检测的最先进方法广泛采用(例如，[3，50])。

https://github.com/baolingfeng/SANER2018Sandboxes

https://github.com/baolingfeng/SANER2018Sandboxes

Table 6: DSM vs. Baselines."APR" stands for "Approach", "TP" is True Positives, "FN" is False Negatives, "TN" is True Negatives, "FP" is False Positives, "TPR" is "True Positive Rate", and "FPR" is "False Positive Rate".

表 6:需求侧管理与基线。“APR”代表“方法”，“TP”代表真阳性，“FN”代表假阴性，“TN”代表真阴性，“FP”代表假阳性，“TPR”代表“真阳性率”，“FPR”代表“假阳性率”。

APR TP FN TN FP TPR FPR

FPR

DSM 93 9 398 112 91.18% 21.97%

需求侧管理 93 9 398 112 91.18% 21.97%

k-tails 94 8 350 160 92.16% 31.37%

k-tails 94 8 350 160 92.16% 31.37%

Boxmate 77 25 421 89 75.49% 17.45%

Boxmate 77 25 421 89 75.49% 17.45%

To compute True Positives and False Negatives, for each pair of benign and malicious app we employ DSM, k-tails, and Boxmate to construct behavior models using training traces of the benign apps, and deploy these models on traces of the corresponding malicious app.Following Equation 2, we use the values of True Positives and False Negatives to estimate True Positive Rates (TPR).The higher the values of TPR, the more malicious activities are detected.On the other hand, to calculate True Negatives and False Positives, for each benign Android app we perform cross-validation among the five test case generation tools (i.e., Monkey [1], GUIRipper [2], PUMA [18], Droidmate [22], and Droidbot [30]).In particular, we use execution traces of 4 tools as training data to learn behavior models by DSM, k-tails and Boxmate.Then, we deploy the inferred models on traces of the remaining tool to check whether the models detect benign apps as malicious (i.e., false positive).In total, we analyze 5 × 102 = 510 combinations between benign apps and test case generation tools.Then, we utilize values of True Negatives and False Positives to compute False Positive Rate (FPR) (see Equation 3).The smaller the values of FPR, the smaller the number of false alarms.

为了计算真阳性和假阴性，对于每对良性和恶意应用，我们使用 DSM、k-tails 和 Boxmate 来构建行为模型，使用良性应用的训练轨迹，并将这些模型部署在相应恶意应用的轨迹上。根据等式 2，我们使用真阳性和假阴性的值来估计真阳性率(TPR)。TPR 值越高，检测到的恶意活动就越多。另一方面，为了计算真阴性和假阳性，对于每一个良性的安卓应用程序，我们在五个测试用例生成工具中执行交叉验证(即，猴子[1、桂利普[2、彪马[18、机器人[22 和机器人[30])。特别是，我们使用 4 个工具的执行轨迹作为训练数据，学习 DSM、k-tails 和 Boxmate 的行为模型。然后，我们在剩余工具的踪迹上部署推断的模型，以检查模型是否将良性应用检测为恶意应用(即假阳性)。我们总共分析了良性应用和测试用例生成工具之间的 5 × 102 = 510 个组合。然后，我们利用真阴性和假阳性的值来计算假阳性率(FPR)(见等式 3)。FPR 值越小，错误警报的数量就越少。

Table 6 shows results for DSM, k-tails, and Boxmate.Boxmate can detect 75.49% of the malicious apps as such while suffering a false positive rate of 17.43%.We note that DSM outperforms Boxmate in terms of True Positive Rate by 15.69% while only losing in terms of False Positive Rate by 4.52%.Comparing DSM with k-tails, we note that they have similar True Positive Rate (difference is less than 1%), but the latter has substantially higher False Positive Rate (difference is close to 10%).Clearly, DSM achieves the best trade-off considering both True Positive Rate and False Positive Rate.

表 6 显示了 DSM、k-tails 和 Boxmate 的结果。Boxmate 可以检测到 75.49%的恶意应用程序，而误报率为 17.43%。我们注意到，DSM 在真阳性率方面的表现优于 Boxsmate 15.69%，而在假阳性率方面仅损失 4.52%。将需求侧管理与 k 尾进行比较，我们注意到它们具有相似的真阳性率(差异小于 1%)，但后者具有显著更高的假阳性率(差异接近 10%)。显然，需求侧管理在考虑真阳性率和假阳性率的情况下达到了最佳的折衷。

6 THREATS TO VALIDITY

6 有效性威胁

Threats to internal validity.We have carefully checked our im-plementation, but there are errors that we did not notice.There are also potential threats related to correctness of ground truth models created by Krka et al. [24] that we used.To mitigate this threat, we have compared their models against execution traces collected from Randoop generated test cases as well as textual documentations published by library class writers (e.g., Javadocs).We revised the ground truth models accordingly.

对内部有效性的威胁。我们已经仔细检查了我们的实施情况，但是有些错误我们没有注意到。我们使用的由克尔卡等人([24)创建的基本真理模型的正确性也有潜在的威胁。为了减轻这种威胁，我们将它们的模型与从 Randoop 生成的测试用例以及由库类作者(例如 Javadocs)发布的文本文档中收集的执行跟踪进行了比较。我们相应地修改了基本事实模型。

Another threat to validity is related to parameter values of tar-get API methods.We use traces collected by Bao et al. [4] which exclude all parameter values.This is different from Jamrozik et al.'s

有效性的另一个威胁与 tar-get API 方法的参数值有关。我们使用包等人[4]收集的轨迹，这些轨迹排除了所有参数值。这与 Jamrozik 等人的不同

This is inline with results that were reported in Boxmate's original paper [21].They reported that for the 18 use cases considered, two false alarms were raised (see Table 2 of their paper) - this results in a False Positive Rate of 11.11%.In our experiment, we consider many more use cases with the aid of different test case generation tools.Jamrozik et al. did not report true positive rate since no malicious apps are considered in the study.

这与博克斯马特的原始论文《[21》中报道的结果是一致的。他们报告说，在考虑的 18 个用例中，出现了两个错误警报(见他们论文的表 2)-这导致了 11.11%的错误阳性率。在我们的实验中，我们借助不同的测试用例生成工具来考虑更多的用例。Jamrozik 等人没有报告真正的阳性率，因为研究中没有考虑恶意应用。

114

work [21] that excludes most (but not all) parameter values.We decide to exclude all parameter values since all specification mining algorithms considered in this paper (including DSM) produce FSAs that have no constraints on values of parameters.As future work, we plan to extend DSM to generate models that include constraints on parameter values.

工作[21]排除大多数(但不是全部)参数值。我们决定排除所有参数值，因为本文中考虑的所有规范挖掘算法(包括需求侧管理)都会产生对参数值没有约束的 FSA。作为未来的工作，我们计划扩展需求侧管理以生成包含参数值约束的模型。

Threats to External Validity.These threats correspond to the generalizability of our empirical findings.In this work, we have analyzed 11 different library classes.This is larger than the number of target classes used to evaluate many prior studies, e.g., [24, 33, 34].As future works, we plan to reduce this threat by analyzing more library classes to infer their automaton based specifications.

对外部有效性的威胁。这些威胁对应于我们经验发现的普遍性。在这项工作中，我们分析了 11 个不同的图书馆类。这大于用于评估许多先前研究的目标类别的数量，例如[24、33、34]。作为未来的工作，我们计划通过分析更多的库类来推断它们基于自动机的规范，从而减少这种威胁。

Threats to Construct Validity.These threats correspond to the usage of evaluation metrics.We have followed Lo and Khoo's ap-proach that uses precision, recall, and F-measure to measure the accuracy of automata output by a specification mining algorithm against ground truth models

[32].Furthermore, Lo and Khoo's approach is well known and has been adopted by many previous re-search works in specification mining e.g., [5, 6, 8, 13, 24, 27, 31, 33].Additionally, True Positive Rate and False Positive Rate are well-known metrics and widely adopted by state-of-the-art approaches in Android malware detection (e.g., [3, 50]).

对结构有效性的威胁。这些威胁对应于评估指标的使用。我们遵循了劳和霍的方法，该方法使用精确度、召回率和离差度量来测量自动机输出的精确度，该自动机输出是通过规范挖掘算法根据基本事实模型[32]进行的。此外，Lo 和 Khoo 的方法是众所周知的，并且已经被规范挖掘中的许多先前的搜索工作所采用，例如，[5、6、8、13、24、27、31、33]。此外，真阳性率和假阳性率是众所周知的指标，并被安卓恶意软件检测领域的最先进方法(例如[3，50)广泛采用。

7 RELATED WORK

7 相关工作

Mining Specifications.Aside from the state-of-the-art baselines considered in Section 4, there are other related works that mine FSA-based specifications from execution traces.Lo et al. propose SMArTIC that mines a FSA from a set of execution traces [13] using a variant of k-tails algorithm that constructs a probabilistic FSA.Mariani et al. propose k-behavior [36] that constructs an automaton by analyzing one single trace at a time.Walkinshaw and Bogdanov propose an approach that allows users to input temporal properties to support a specification miner to construct a FSA from execution traces [45].Lo et al. further extend Walkinshaw and Bogdanov's work to automatically infer temporal properties from execution traces, and use these properties to automatically support model inference process of a specification miner [33].Synoptic infers three kinds of temporal invariants from execution traces and uses them to generate a concise FSA [6].SpecForge [26] is a meta-approach that analyzes FSAs inferred by other specification miners and combine them together to create a more accurate FSA.None of the above mentioned approaches employs deep learning.

采矿规范。除了第 4 节中考虑的最先进的基线之外，还有从执行跟踪中挖掘基于 FSA 的规范的其他相关工作。Lo 等人提出了智能集成电路，该智能集成电路使用构造概率性有限状态机的 k-tails 算法的变体，从一组执行轨迹中挖掘有限状态机。马里亚尼等人提出了 k 行为[36]，它通过一次分析一条轨迹来构造一个自动机。沃金肖和博格丹诺夫提出了一种方法，允许用户输入时间属性，以支持规范挖掘器根据执行轨迹构建一个 FSA[45]。Lo 等人进一步扩展了 Walkinshaw 和 Bogdanov 的工作，从执行轨迹中自动推断时间属性，并使用这些属性来自动支持规范挖掘者[33]的模型推断过程。概要从执行轨迹中推断出三种时间不变量，并使用它们生成一个简明的 FSA [6]。SpecForge [26]是一种元方法，它分析由其他规范挖掘者推断的金融服务协议，并将它们组合在一起，以创建更准确的金融服务协议。上述方法都没有采用深度学习。

Deep Learning for Software Engineering Tasks.Recently, deep learning methods are proposed to learn representations of data with multiple levels of abstraction [28].Researchers have been utilizing deep learning to solve challenging tasks in software engi-neering [17, 25, 4749].For example, Gu et al. propose DeepAPI that takes as input queries in natural languages and outputs sequences of API methods that developers should follow [17].In the nutshell, DeepAPI replies on a RNN based model that can translate a sentence in one language to a new sentence in another language.Different from DeepAPI, DSM takes as input method sequences of an API or library and outputs a finite-state automaton that represents behav-iors of that API or library.Prior to our work, deep learning models have not been employed to effectively mine specifications.

软件工程任务的深度学习。最近，提出了深度学习方法来学习具有多个抽象层次的数据表示[28]。研究人员一直在[17，25，4749]利用深度学习解决软件工程中的挑战性任务。例如，Gu 等人提出了 DeepAPI，它将自然语言中的查询作为输入，并输出开发人员应该遵循[17]的 API 方法序列。简而言之，

DeepAPI 回复了一个基于 RNN 的模型，该模型可以将一种语言的句子翻译成另一种语言的新句子。与深度应用编程接口不同，需求侧管理将应用编程接口或库的输入方法序列作为输入，并输出表示该应用编程接口或库行为的有限状态自动机。在我们工作之前，深度学习模型还没有被用来有效地挖掘规格。

Language Models for Software Engineering Tasks.Statistical language models have been utilized for many software engineering tasks.For example, Hindle et al. employ n-gram model on code tokens to demonstrate a high degree of local repetitiveness in source code corpora and leverage it to improve Eclipse's code completion engine [19].Several other works have extended Hindle et al. work to build more powerful code completion engines;for example, Ray-chev et al. leverage n-gram and recurrent neural network language model to recommend likely sequences of method calls to a program with holes [41], while Nguyen et al. leverage Hidden Markov Model to learn API usages from Android app bytecode for recommending APIs [40].Beyond code completion, Wang et al. use n-gram model to detect bugs by identifying low probability token sequences [46].Our work uses language model for a different task.

软件工程任务的语言模型。统计语言模型已经被用于许多软件工程任务。例如，欣德尔等人在代码库中使用 n-gram 模型来证明源代码语料库的高度局部重复性，并利用它来改进 Eclipse 的代码完成引擎[19]。其他几项工作扩展了 Hindle 等人的工作，以构建更强大的代码完成引擎；例如，雷-切夫等人利用 n-gram 和递归神经网络语言模型向具有漏洞的程序推荐可能的方法调用序列[41]，而阮氏等人利用隐马尔可夫模型从安卓应用程序字节码中学习应用程序接口用法以推荐应用程序接口[40]。除了代码完成之外，王等人使用 n-gram 模型通过识别低概率令牌序列来检测错误[46]。我们的工作使用语言模型来完成不同的任务。

8 CONCLUSION AND FUTURE WORK

8 结论和未来工作

Formal specifications are helpful for many software processes.In this work, we propose DSM, a new approach that employs Recur-rent Neural Network Based Language Model (RNNLM) for mining FSA-based specifications.We apply Randoop, a well-known test cases generation approach, to create a richer set of execution traces for training RNNLM.From a set of sampled execution traces, we construct a Prefix Tree Acceptor (PTA) and extract many features of PTA's states using the learned RNNLM.These features are then utilized by clustering algorithms to merge similar automata states to construct many FSAs using various settings.Then, we employ a model selection heuristic to select the FSA that is estimated to be the most accurate and output it as the final model.We run our proposed approach to infer specifications of 11 target library classes.Our results show that DSM achieves an average F-measure of 71.97%, outperforming the best performing baseline by 28.82%.Addition-ally, we propose a technique that employs a FSA mined by DSM to detect malicious behaviors in an Android app.In particular, our technique uses the inferred FSA as a behavior model to construct a more comprehensive sandbox that considers execution context of sensitive API methods.Our evaluation shows that the proposed technique can improve the True Positive Rate of Boxmate by 15.69% while only increasing the False Positive Rate by 4.52%.

正式规范对许多软件过程都有帮助。在这项工作中，我们提出了需求侧管理，这是一种利用基于递归神经网络的语言模型来挖掘基于需求侧管理的规范的新方法。我们应用 Randoop，一种众所周知的测试用例生成方法，为训练 RNNLM 创建一组更丰富的执行跟踪。从一组采样的执行轨迹中，我们构造了一个前缀树接受器(PTA)，并利用学习的 RNNLM 提取了 PTA 状态的许多特征。然后，聚类算法利用这些特征来合并相似的自动机状态，以使用各种设置构建许多 FSa。然后，我们使用模型选择启发式算法来选择被估计为最精确的有限状态机，并将其作为最终模型输出。我们运行我们提出的方法来推断 11 个目标库类的规范。我们的结果显示，需求侧管理的平均离差为 71.97%，比最佳绩效基线高出 28.82%。此外，我们还提出了一种技术，该技术采用了由需求侧管理挖掘的前端服务协议来检测安卓应用程序中的恶意行为。特别是，我们的技术使用推断的 FSA 作为行为模型来构建一个更全面的沙箱，该沙箱考虑敏感的

应用编程接口方法的执行上下文。我们的评估表明，该技术可将黄杨的真阳性率提高 15.69%，而假阳性率仅提高 4.52%。

As future work, we plan to improve DSM's effectiveness further by integrating information of likely invariants into our deep learn-ing based framework.We also plan to employ EvoSuite [16] and many other test case generation tools to generate an even more comprehensive set of training traces to improve the effectiveness of DSM.Furthermore, we plan to improve DSM by considering many more clustering algorithms aside from k-means and hierarchical clustering, especially the ones that require no inputs for the number of clusters (e.g., DBSCAN [15], etc.).Finally, we plan to evaluate DSM with more classes and libraries in order to reduce threats to external validity.

作为未来的工作，我们计划通过将可能不变量的信息集成到我们基于深度学习的框架中来进一步提高需求侧管理的有效性。我们还计划使用 EvoSuite [16]和许多其他测试用例生成工具来生成一组更加全面的培训跟踪，以提高需求侧管理的有效性。此外，我们计划通过考虑除 k 均值和分层聚类之外的更多聚类算法来改进需求侧管理，尤其是那些不需要输入聚类数量的算法(例如，DBSCAN [15)，等等。)。最后，我们计划用更多的类和库来评估需求侧管理，以减少对外部有效性的威胁。

Additional Resources and Acknowledgement.Our dataset, DSM's implementation, and a technical report that includes additional results are publicly available at: https://github.com/lebuitienduy/DSM.This research was supported by the Singapore National Research Foundations National Cybersecurity Research & Development Programme (award number: NRF2016NCR-NCR001- 008).

附加资源和确认。我们的数据集、电力需求侧管理的实施以及包含其他结果的技术报告可在以下网站上公开获取:https://github.com/电力需求侧管理。这项研究得到了新加坡国家研究基金会国家网络安全研究与发展计划的支持(获奖号码:NRF2016NCR-NCR001- 008)。

115

REFERENCES

参考

[1] Last accessed on February 25, 2017.In https://github.com/linkedin/camus.

[1]最后一次访问是在 2017 年 2 月 25 日。在 https://github.com/linkedin/camus.

[2] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon.2012.Using GUI ripping for automated testing of Android applications.In IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012.258-261.

[2]多梅尼科·阿马尔菲塔诺、安娜·丽塔·法索里诺、波菲里奥·特拉蒙塔纳、萨尔瓦托勒·德·卡明和阿蒂夫·梅蒙。2012.使用图形用户界面抓取来自动测试安卓应用程序。在 2012 年 9 月 3 日至 7 日于德国埃森举行的美国电气工程师协会/美国计算机学会自动化软件工程国际会议上。258-261。

[3] Daniel Arp, Michael Spreitzenbarth, Malte Hubner, Hugo Gascon, and Konrad Rieck.2014.DREBIN: Effective and Explainable Detection of Android Malware in Your Pocket.In 21st Annual Network and Distributed System Security Symposium, NDSS 2014, San Diego, California, USA, February 23-26, 2014.

[3]丹尼尔·阿尔普、迈克尔·斯普雷岑巴特、马尔特·胡布纳、雨果·加斯孔和康拉德·里克。2014.DREBIN:在你的口袋里有效和可解释的检测安卓恶意软件。第 21 届年度网络和分布式系统安全研讨会，2014 年 NDSS，美国加利福尼亚州圣地亚哥，2014 年 2 月 23 日至 26 日。

[4] Lingfeng Bao, Tien-Duy B. Le, and David Lo.2018.Mining sandboxes: Are we there yet?.In 25th International Conference on Software Analysis, Evolution and Reengineering, SANER 2018, Campobasso, Italy, March 20-23, 2018.445-455.

[4]灵峰宝、天都乐、卢大伟。2018.采矿沙箱:我们到了吗？。2018 年 3 月 20 日至 23 日在意大利坎波巴索举行的第 25 届软件分析、进化和再造国际会议上。445-455。

[5] Ivan Beschastnikh, Yuriy Brun, Jenny Abrahamson, Michael D. Ernst, and Arvind Krishnamurthy.2015.Using Declarative Specification to Improve the Under-standing, Extensibility, and Comparison of Model-Inference Algorithms.IEEE Trans.Software Eng.41, 4 (2015), 408-428.

[5]伊万·贝沙斯蒂尼克、塞维多夫·布伦、珍妮·亚伯拉罕森、迈克尔·恩斯特和阿尔温德·克里希纳穆西。2015.使用声明性规范来改进模型推理算法的不足、可扩展性和比较。IEEE 传输。软件工程。41，4 (2015)，408-428。

[6] Ivan Beschastnikh, Yuriy Brun, Sigurd Schneider, Michael Sloan, and Michael D Ernst.2011.Leveraging existing instrumentation to automatically infer invariant-constrained models.In Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering.ACM, 267-277.

[6]伊万·贝沙斯蒂尼克、塞维多夫·布伦、齐格鲁德·施耐德、迈克尔·斯隆和迈克尔·恩斯特。2011.利用现有仪器自动推断不变约束模型。在第 19 届 ACM SIGSOFT 研讨会和第 13 届欧洲软件工程基础会议记录中。ACM，267-277。

[7] Alan W Biermann and Jerome A Feldman.1972.On the synthesis of finite-state machines from samples of their behavior.IEEE Trans.Comput.100, 6 (1972), 592-597.

[7]艾伦·比尔曼和杰罗姆·费尔德曼。1972.从有限状态机的行为样本合成有限状态机。IEEE 传输。计算机。100，6 (1972)，592-597。

[8] Zherui Cao, Yuan Tian, Tien-Duy B Le, and David Lo.[n. d.].Rule-based specifi-cation mining leveraging learning to rank.Automated Software Engineering ([n. d.]), 1-30.

[8]曹哲瑞、田园、天都乐、卢大伟。[特区]。基于规则的指定挖掘利用学习排名。自动化软件工程([特区)，1-30。

[9] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio.2014.Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling.CoRR abs/1412.3555 (2014).

[9]钟俊英、恰拉尔·居莱尔、赵庆云和约绍·本吉奥。2014.门控递归神经网络对序列建模的实证评价。CoRR abs/1412.3555 (2014)。

[10] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled.1999.Model Checking.MIT Press, Cambridge, MA, USA.

[10]小埃德蒙·克拉克、阿娜·格鲁伯格和多伦·佩莱德。1999.模型检查。麻省理工学院出版社，美国马萨诸塞州剑桥。

[11] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Gordon Fraser, Sebas-tian Hack, and Andreas Zeller.2012.Automatically Generating Test Cases for Specification Mining.IEEE Trans.Software Eng.38, 2 (2012), 243-257.

[11]瓦伦汀·达尔梅尔、尼古拉·克诺普、克里斯托夫·马龙、戈登·弗雷泽、赛巴斯·田哈克和安德列亚斯·泽勒。2012.为规范挖掘自动生成测试用例。IEEE 传输。软件工程。38，2 (2012)，243-257。

[12] Valentin Dallmeier, Nikolai Knopp, Christoph Mallon, Sebastian Hack, and An-dreas Zeller.2010.Generating test cases for specification mining.In Proceedings of the Nineteenth International Symposium on Software Testing and Analysis, ISSTA 2010, Trento, Italy, July 12-16, 2010.85-96.

[12]瓦伦汀·达尔梅尔、尼古拉·克诺普、克里斯托夫·马龙、塞巴斯蒂安·哈克和安-德雷斯·泽勒。2010.为规范挖掘生成测试用例。2010 年 7 月 12 日至 16 日在意大利特伦托举行的国际软件测试与分析研讨会论文集。85-96。

[13] David Lo and Siau-Cheng Khoo.2006.SMArTIC: towards building an accu-rate, robust and scalable specification miner.In Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2006, Portland, Oregon, USA, November 5-11, 2006.265-275.

[13]卢大伟和肖成浩。2006.SMArTIC:致力于构建一个精确、健壮和可扩展的规范挖掘器。2006 年 11 月 5 日至 11 日在美国俄勒冈州波特兰市举行的第 14 届美国计算机学会软件工程基础国际研讨会记录。265-275。

[14] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao.2007.The Daikon system for dynamic detection of likely invariants.Sci.Comput.Program.69, 1-3 (2007), 35-45.

[14]迈克尔·恩斯特、杰夫·珀金斯、菲利普·郭、斯蒂芬·麦卡曼、卡洛斯·帕切科、马修·茨尚茨和陈晓。2007.动态检测可能不变量的 Daikon 系统。Sci。计算机。程序。69，1-3 (2007)，35-45。

[15] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu.1996.A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise.In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA.226-231.

[15]马丁·埃斯特、汉斯-彼得·克里格尔、约格·桑德和徐小薇。1996.一种基于密度的大型噪声空间数据库聚类发现算法。在美国俄勒冈州波特兰举行的第二届知识发现和数据挖掘国际会议(KDD-96)上。226-231。

[16] Gordon Fraser and Andrea Arcuri.2011.EvoSuite: automatic test suite generation for object-oriented software.In SIGSOFT/FSE'11 19th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-19) and ESEC'11: 13th European Software Engineering Conference (ESEC-13), Szeged, Hungary, September 5-9, 2011.416-419.

[16]戈登·弗雷泽和安德里亚·阿库里。2011.EvoSuite:面向对象软件的自动测试套件生成。在 2011 年 9 月 5 日至 9 日在匈牙利塞格德举行的 SIGNSOFT/FSE 第 11 届第 19 届 ACM SIGSOFT 软件工程基础研讨会(FSE-19)和 ESEC 第 11 届第 13 届欧洲软件工程会议(ESEC-13)上。416-419。

[17] Xiaodong Gu, Hongyu Zhang, Dongmei Zhang, and Sunghun Kim.2016.Deep API learning.In Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016.631-642.

[17]顾晓东、张洪宇、张冬梅和金成勋。2016.深度应用编程接口学习。2016 年 11 月 13 日至 18 日在美国华盛顿州西雅图市举行的第 24 届美国计算机学会软件工程基础国际研讨会记录。631-642。

[18] Shuai Hao, Bin Liu, Suman Nath, William G. J. Halfond, and Ramesh Govindan.2014.PUMA: programmable UI-automation for large-scale dynamic analysis of mobile apps.In The 12th Annual International Conference on Mobile Systems, Applications, and Services, MobiSys'14, Bretton Woods, NH, USA, June 16-19, 2014.204-217.

[18]郝帅、柳斌、苏曼·纳特、威廉·哈尔丰德和拉梅什·戈文丹。2014.PUMA:可编程用户界面自动化，用于移动应用的大规模动态分析。2014 年 6 月 16 日至 19 日，在美国新罕布什尔州布雷顿森林举行的第 12 届移动系统、应用和服务国际年会上。204-217。

[19] Abram Hindle, Earl T. Barr, Zhendong Su, Mark Gabel, and Premkumar T. De-vanbu.2012.On the naturalness of software.In 34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland.837-847.

[19]亚伯兰·欣德尔、厄尔·巴尔、苏振东、马克·加贝尔和普雷姆库马·德·万布。2012.论软件的自然性。在 2012 年 6 月 2 日至 9 日于瑞士苏黎士举行的第 34 届软件工程国际会议上，2012 年，ICSE。837-847。

[20] Sepp Hochreiter and Jürgen Schmidhuber.1997.Long short-term memory.Neural computation 9, 8 (1997), 1735-1780.

[20]塞普·霍克莱特和于尔根·施密德胡伯。1997.长期短期记忆。神经计算 9，8 (1997)，1735-1780。

[21] Konrad Jamrozik, Philipp von Styp-Rekowsky, and Andreas Zeller.2016.Min-ing sandboxes.In Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016.37-48.

[21]康拉德·贾姆罗齐克、菲利普·冯·斯蒂普-雷考斯基和安德烈亚斯·泽勒。2016.最小沙箱。2016 年 5 月 14 日至 22 日在美国德克萨斯州奥斯汀市 ICSE 举行的第 38 届国际软件工程会议记录。37-48。

[22] Konrad Jamrozik and Andreas Zeller.2016.DroidMate: a robust and extensible test generator for Android.In Proceedings of the International Conference on Mobile Software Engineering and Systems, MOBILESoft '16, Austin, Texas, USA, May 14-22, 2016.293-294.

[22]康拉德·贾姆罗齐克和安德烈亚斯·泽勒。2016.机器人(DroidMate):一个健壮且可扩展的安卓测试生成器。在 2016 年 5 月 14 日至 22 日在美国德克萨斯州奥斯汀举行的移动软件工程和系统国际会议记录中。293-294。

[23] John C. Knight, Colleen L. DeJong, Matthew S. Gibble, and LuÃŋs G. Nakano.1997.Why Are Formal Methods Not Used More Widely?.In Fourth NASA Formal Methods Workshop.1-12.

约翰·奈特、科琳·德容、马修·吉布和 luãŋs·中野 1997.为什么正式方法没有得到更广泛的应用？。美国宇航局第四次正式方法研讨会。1-12。

[24] Ivo Krka, Yuriy Brun, and Nenad Medvidovic.2014.Automatic mining of specifi-cations from invocation traces and method invariants.In Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014.178-189.

[24]伊沃·克尔卡、塞维多夫·布伦和内纳德·梅德维多维奇。2014.从调用跟踪和方法不变量中自动挖掘指定。2014 年 11 月 16 日至 22 日在中国香港举行的第 22 届美国计算机学会软件工程基础国际研讨会论文集(FSE-22)。178-189。

[25] An Ngoc Lam, Anh Tuan Nguyen, Hoan Anh Nguyen, and Tien N. Nguyen.2015.Combining Deep Learning with Information Retrieval to Localize Buggy Files for Bug Reports (N).In 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015.476-481.

[25]安哥林、安团阮、和安阮和田恩阮。2015.将深度学习与信息检索相结合，为错误报告本地化错误文件。2015 年 11 月 9 日至 13 日在美国东北林肯举行的第 30 届美国电气工程师协会/美国计算机学会自动化软件工程国际会议上。476-481。

[26] Tien-Duy B. Le, Xuan-Bach D. Le, David Lo, and Ivan Beschastnikh.2015.Syn-ergizing Specification Miners through Model Fissions and Fusions (T).In 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, Lincoln, NE, USA, November 9-13, 2015.115-125.

[26]蒂恩-杜伊·贝尔、轩-巴赫·戴尔、卢大伟和伊万·贝斯恰斯蒂尼克。2015.通过模型裂变和聚变给规范矿工提供能量。2015 年 11 月 9 日至 13 日在美国东北林肯举行的第 30 届美国电气工程师协会/美国计算机学会自动化软件工程国际会议上。115-125。

[27] Tien-Duy B. Le and David Lo.2015.Beyond support and confidence: Exploring interestingness measures for rule-based specification mining.In 22nd IEEE Inter-national Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015.331-340.

[27]蒂恩-杜伊·贝尔和卢大伟。2015.超越支持和信任:探索基于规则的规范挖掘的兴趣度度量。2015 年 3 月 2 日至 6 日在加拿大蒙特利尔举行的第 22 届美国电气工程师学会软件分析、进化和再设计国际会议上。331-340。

[28] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton.2015.Deep learning.Nature 521, 7553 (2015), 436-444.

[28]扬·勒春、约舒·本吉奥和杰弗里·辛顿。2015.深度学习。自然 521，7553 (2015)，436-444。

[29] Li Li, Daoyuan Li, Tegawendé F. Bissyandé, Jacques Klein, Yves Le Traon, David Lo, and Lorenzo Cavallaro.2017.Understanding Android App Piggybacking: A Systematic Study of Malicious Code Grafting.IEEE Trans.Information Forensics and Security 12, 6 (2017), 1269-1284.

[29]李莉、李道元、泰加文代夫·比斯安代、雅克·克莱因、伊夫·勒特龙、卢大伟和洛伦佐·卡瓦拉罗。2017.理解安卓应用搭载:恶意代码移植的系统研究。IEEE 传输。信息取证与安全 12，6 (2017)，1269-1284。

[30] Yuanchun Li, Ziyue Yang, Yao Guo, and Xiangqun Chen.2017.DroidBot: a lightweight UI-guided test input generator for Android.In Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume.23-26.

[30]李园春、杨子岳、郭尧和陈向群。2017.机器人(DroidBot):一个面向安卓的轻量级用户界面引导的测试输入生成器。在 2017 年 ICSE 第 39 届国际软件工程会议记录中，阿根廷布宜诺斯艾利斯，2017 年 5 月 20-28 日-配套卷。23-26。

[31] David Lo, Hong Cheng, Jiawei Han, Siau-Cheng Khoo, and Chengnian Sun.2009.Classification of software behaviors for failure detection: a discriminative pattern mining approach.In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining.ACM, 557-566.

[31]卢大伟、洪成、韩嘉伟、肖成浩、孙成年。2009.用于故障检测的软件行为分类:一种判别模式挖掘方法。在第 15 届 ACM 全球知识发现与数据挖掘国际会议记录中。ACM，557-566。

[32] David Lo and Siau-Cheng Khoo.2006.QUARK: Empirical Assessment of Automaton-based Specification Miners.In 13th Working Conference on Reverse Engineering (WCRE 2006), 23-27 October 2006, Benevento, Italy.51-60.

[32]卢大伟和肖成浩。2006.夸克:基于自动机的规范挖掘器的经验评估。第十三届逆向工程工作会议(2006 年，WCRE)，2006 年 10 月 23 日至 27 日，意大利贝内文托。51-60。

[33] David Lo, Leonardo Mariani, and Mauro Pezzè.2009.Automatic steering of behavioral model inference.In Proceedings of the 7th joint meeting of the European Software Engineering Conference and the ACM SIGSOFT International Symposium on Foundations of Software Engineering, 2009, Amsterdam, The Netherlands, August 24-28, 2009.345-354.

[33]卢大伟、莱昂纳多·马里亚尼和毛罗·佩泽。2009.行为模型推理的自动导向。2009 年 8 月 24 日至 28 日在荷兰阿姆斯特丹举行的欧洲软件工程会议和 ACM SIGSOFT 软件工程基础国际研讨会第七届联席会议记录。345-354。

[34] David Lo, Leonardo Mariani, and Mauro Santoro.2012.Learning extended FSA from software: An empirical assessment.Journal of Systems and Software 85, 9 (2012), 2063-2076.

[34]卢大伟、莱昂纳多·马里亚尼和毛罗·桑托罗。2012.从软件中学习扩展的 FSA:一个经验评估。系统和软件杂志 85，9 (2012)，2063-2076。

[35] James MacQueen et al. 1967.Some methods for classification and analysis of multivariate observations.In Proceedings of the fifth Berkeley symposium on mathematical statistics and probability, Vol. 1.Oakland, CA, USA., 281-297.

[35]詹姆斯·麦克奎恩等人，1967 年。多元观测数据分类和分析的一些方法。在第五届伯克利数理统计和概率研讨会记录，第 1 卷。美国加利福尼亚州奥克兰。，281-297。

[36] Leonardo Mariani, Fabrizio Pastore, and Mauro Pezzè.2011.Dynamic Analysis for Diagnosing Integration Faults.IEEE Trans.Software Eng.37, 4 (2011), 486-508.

[36]莱昂纳多·马里亚尼，法布里齐奥·帕斯托雷和毛罗·佩泽。2011.集成故障诊断的动态分析。IEEE 传输。软件工程。37，4 (2011)，486-508。

[37] Weikai Miao and Shaoying Liu.2012.A Formal Specification-Based Integration Testing Approach.In SOFL.26-43.

[37]魏凯苗和应劭刘。2012.一种基于规范的集成测试方法。在 SOFL。26-43。

[38] Tomas Mikolov, Anoop Deoras, Stefan Kombrink, Lukás Burget, and Jan Cernocký.2011.Empirical Evaluation and Combination of Advanced Language Modeling Techniques.In INTERSPEECH 2011, 12th Annual Conference of the In-ternational Speech Communication Association, Florence, Italy, August 27-31, 2011.605-608.

[38]托马斯·米科洛夫、阿诺普·德奥拉斯、斯特凡·康布林克、卢克斯·勃艮第和扬·瑟-诺克。2011.高级语言建模技术的实证评估和组合。2011 年 8 月 27 日至 31 日在意大利佛罗伦萨举行的国际语言交流协会第十二届年会。605-608。

[39] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černocky, and Sanjeev Khu- ` danpur.2010.Recurrent Neural Network Based Language Model.In Eleventh Annual Conference of the International Speech Communication Association.

[39]托马什·米科洛夫、马丁·卡拉菲埃特、卢克·勃艮第、扬·恩诺基和桑吉夫·库-`丹普尔。2010.基于递归神经网络的语言模型。在国际言语交际协会第十一届年会上。

[40] Tam The Nguyen, Hung Viet Pham, Phong Minh Vu, and Tung Thanh Nguyen.2016.Learning API usages from bytecode: a statistical approach.In Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016.416-427.

[40]阮潭、洪越、冯明武和阮东三。2016.从字节码中学习应用编程接口用法:一种统计方法。2016 年 5 月 14 日至 22 日在美国德克萨斯州奥斯汀市 ICSE 举行的第 38 届国际软件工程会议记录。416-427。

[41] Veselin Raychev, Martin T. Vechev, and Eran Yahav.2014.Code completion with statistical language models.In ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '14, Edinburgh, United Kingdom - June 09 - 11, 2014.419-428.

[41]维塞林·赖切夫、马丁·维切夫和埃兰·亚哈夫。2014.用统计语言模型完成代码。2014 年 6 月 9 日至 11 日，在英国爱丁堡 PLDI 14 日举行的 ACM SIGPLAN 编程语言设计与实现会议上。419-428。

[42] Brian Robinson, Michael D. Ernst, Jeff H. Perkins, Vinay Augustine, and Nuo Li.2011.Scaling up automated test generation: Automatically generating main-tainable regression unit tests for programs.In ASE 2011: Proceedings of the 26th Annual International Conference on Automated Software Engineering.Lawrence, KS, USA, 23-32.

[42]布赖恩·罗宾逊、迈克尔·恩斯特、杰夫·珀金斯、维奈·奥古斯丁和李诺。2011.扩展自动测试生成:为程序自动生成可维护的回归单元测试。ASE 2011:第 26 届自动化软件工程国际年会论文集。美国堪萨斯州劳伦斯市，23-32。

[43] Lior Rokach and Oded Maimon.2005.Clustering Methods.In The Data Mining and Knowledge Discovery Handbook.321-352.

[43]利奥·洛克赫和奥德·梅蒙。2005.聚类方法。数据挖掘和知识发现手册。321-352。

[44] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le.2014.Sequence to Sequence Learning with Neural Networks.In Advances in Neural Information Processing Systems 27: Annual Conference on

Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada.3104-3112.

[44]伊利亚·苏斯基弗，奥里奥尔·温耶尔斯和科克·维尔。2014.用神经网络进行序列到序列学习。《神经信息处理系统的进展》第 27 期:2014 年神经信息处理系统年会，2014 年 12 月 8 日至 13 日，加拿大魁北克蒙特利尔。3104-3112。

116

[45] Neil Walkinshaw and Kirill Bogdanov.2008.Inferring Finite-State Models with Temporal Constraints.In 23rd IEEE/ACM International Conference on Automated Software Engineering (ASE 2008), 15-19 September 2008, L'Aquila, Italy.248-257.

[45]尼尔·沃金肖和基里尔·博格达诺夫。2008.推断具有时间约束的有限状态模型。2008 年 9 月 15 日至 19 日在意大利拉奎拉举行的第 23 届 IEEE/ACM 自动化软件工程国际会议(ASE 2008)。248-257。

[46] Song Wang, Devin Chollak, Dana Movshovitz-Attias, and Lin Tan.2016.Bugram: bug detection with n-gram language models.In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, ASE 2016, Singapore, September 3-7, 2016.708-719.

[46]王松、德文·乔拉克、达纳·莫夫谢维茨-阿提亚斯和林坦。2016.Bugram:用 n-gram 语言模型进行 bug 检测。2016 年 9 月 3 日至 7 日在新加坡举行的第 31 届美国电气工程师学会/美国计算机学会自动化软件工程国际会议记录。708-719。

[47] Song Wang, Taiyue Liu, and Lin Tan.2016.Automatically learning semantic features for defect prediction.In Proceedings of the 38th International Conference on Software Engineering, ICSE 2016, Austin, TX, USA, May 14-22, 2016.297-308.

[47]王松、刘太岳、林谭。2016.缺陷预测的语义特征自动学习。2016 年 5 月 14 日至 22 日在美国德克萨斯州奥斯汀市 ICSE 举行的第 38 届国际软件工程会议记录。297-308。

[48] Martin White, Michele Tufano, Christopher Vendome, and Denys Poshyvanyk.2016.Deep learning code fragments for code clone detection.In Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering,

[48]马丁·怀特、米歇尔·图法诺、克里斯托弗·文多姆和丹尼斯·波希万尼克。2016.用于代码克隆检测的深度学习代码片段。在第 31 届美国电气工程师学会/美国计算机学会自动化软件工程国际会议记录中，

ASE 2016, Singapore, September 3-7, 2016.87-98.

ASE 2016，新加坡，2016 年 9 月 3-7 日。87-98。

[49] Martin White, Christopher Vendome, Mario Linares Vásquez, and Denys Poshyvanyk.2015.Toward Deep Learning Software Repositories.In 12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015.334-345.

[49]马丁·怀特、克里斯托弗·文多姆、马里奥·利纳雷斯·巴斯克斯和丹尼斯·波希-万尼克。2015.走向深度学习软件仓库。2015 年 5 月 16 日至 17 日在意大利佛罗伦萨举行的第 12 届 IEEE/ACM 采矿软件库工作会议，MSR，2015 年。334-345。

[50] Mu Zhang, Yue Duan, Heng Yin, and Zhiruo Zhao.2014.Semantics-Aware Android Malware Classification Using Weighted Contextual API Dependency Graphs.In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014.1105-1116.

[50]张穆、岳端、恒贤、赵志若。2014.使用加权上下文相关应用编程接口依赖图的语义感知安卓恶意软件分类。2014 年美国亚利桑那州斯科茨代尔 2014 年计算机与通信安全会议录，2014 年 11 月 3-7 日。1105-1116。

[51] Hao Zhong and Zhendong Su.2013.Detecting API documentation errors.In Proceedings of the 2013 ACM SIGPLAN International Conference on Object Oriented Programming Systems Languages & Applications, OOPSLA 2013, part of SPLASH 2013, Indianapolis, IN, USA, October 26-31, 2013.803-816.

[51]钟浩和苏振东。2013.检测应用编程接口文档错误。2013 年美国印第安纳波利斯面向对象编程系统语言与应用国际会议录，面向对象编程语言与应用协会，2013 年，2013 年飞溅的一部分，2013 年 10 月 26-31 日。803-816。

117