# Automatic Mining of Specifications from Invocation Traces and Method Invariants

从调用轨迹和方法不变量中自动挖掘规范

Ivo Krka

Ivo Krka

Google Inc. University of Massachusetts University of Southern California Zurich, Switzerland Amherst, MA, USA Los Angeles, CA, USA

谷歌公司美国马萨诸塞州大学南加州苏黎世大学瑞士阿姆赫斯特分校美国加利福尼亚州洛杉矶

krka@google.com brun@cs.umass.edu neno@usc.edu

krka@google.com·brun@cs.umass.edu·neno@usc.edu

Yuriy Brun Nenad Medvidovic

塞维多夫·布伦·内纳德·梅德维多维奇

## ABSTRACT

## 摘要

Software library documentation often describes individual meth-ods' APIs, but not the intended protocols and method interactions.This can lead to library misuse, and restrict runtime detection of protocol violations and automated verification of software that uses the library.Specification mining, if accurate, can help mitigate these issues, which has led to significant research into new model-inference techniques that produce FSM-based models from program invariants and execution traces.However, there is currently a lack of empirical studies that, in a principled way, measure the impact of the inference strategies on model quality.To this end, we identify four such strategies and systematically study the quality of the models they produce for nine off-the-shelf libraries.We find that (1) using invariants to infer an initial model significantly improves model quality, increasing precision by 4% and recall by 41%, on average;(2) effective invariant filtering is crucial for quality and scalability of strategies that use invariants;and (3) using traces in combina-tion with invariants greatly improves robustness to input noise.We present our empirical evaluation, implement new and extend existing model-inference techniques, and make public our implementations, ground-truth models, and experimental data.Our work can lead to higher-quality model inference, and directly improve the techniques and tools that rely on model inference.Categories and Subject Descriptors:

软件库文档通常描述单个方法的 APIs，但不描述预期的协议和方法交互。这可能导致库误用，并限制协议违规的运行时检测和使用库的软件的自动验证。规范挖掘如果准确的话，可以帮助缓解这些问题，这导致了对新的模型推理技术的重要研究，这些技术从程序不变量和执行轨迹中产生基于状态机的模型。然而，目前缺乏实证研究，以原则性的方式衡量推理策略对模型质量的影响。为此，我们确定了四种这样的策略，并系统地研究了它们为九个现成的图书馆制作的模型的质量。我们发现(1)使用不变量推断初始模型显著提高了模型质量，平均精度提高了 4%，召回率提高了 41%；(2)有效的不变量过滤对于使用不变量的策略的质量和可伸缩性至关重要；以及(3)结合不变量使用迹线极大地提高了对输入噪声的鲁棒性。我们展示我们的经验评估，实现新的和扩展现有的模型推理技术，并公开我们的实现、基本事实模型和实验数据。我们的工作可以导致更高质量的模型推理，并直接改进依赖于模型推理的技术和工具。
类别和主题描述符:

D.2.5 [Testing and Debugging]: Debugging aids, Tracing General Terms: Algorithms, Design, Modeling Keywords: Model inference, execution traces, log analysis

[测试与调试]:调试辅助工具，跟踪通用术语:算法，设计，建模关键词:模型推理，执行跟踪，日志分析

1.INTRODUCTION

1.介绍

Developers frequently use existing software libraries.Unfortu-nately, many libraries are poorly documented, hindering reuse and forcing developers to re-engineer existing solutions [23, 51].Even heavily-used, well-documented libraries contain documentation in-accuracies as updates to the documentation lag or fail to follow This work was completed while Ivo Krka was a PhD student at the University of Southern California.

开发人员经常使用现有的软件库。不幸的是，许多库的文档很少，阻碍了重用，迫使开发人员重新设计现有的解决方案[23，51]。即使是大量使用的、有良好文档记录的图书馆也包含不准确的文档，因为文档更新滞后或未能跟上。这项工作是在伊沃·克尔卡是南加州大学博士生时完成的。



FSE'14, November 16-21, 2014, Hong Kong, China ACM 978-1-4503-3056-5/14/11 http://dx.doi.org/10.1145/2635868.2635890

FSE'14，2014 年 11 月 16-21 日，中国香港 ACM 978-1-4503-3056-5/14/11
http://dx.doi.org/10.1145/2635868.2635890

code updates [14, 54].Further, natural-language documentation can be ambiguous and misunderstood by developers [13], caus-ing library misuses that lead to subtle, latent, and costly faults.A promising way to ease library use is to automatically mine accurate specifications from existing uses of those libraries [59].Numerous specification mining techniques have been proposed [2, 3, 4, 11, 17, 20,21,24,39,41,43,52,58] and used for test case generation [16,41], debugging [4], validation [18], and fault localization [48].

代码更新[14，54]。此外，开发人员[13]可能会对自然语言文档产生歧义和误解，从而导致导致微妙、潜在和代价高昂的错误的库误用。简化库使用的一个有希望的方法是从那些库的现有使用中自动挖掘准确的规范[59]。[2、3、4、11、17、20、21、24、39、41、43、52、58]提出了许多规范挖掘技术，并用于测试用例生成[16、41]、调试[4)、验证[18]和故障定位[48]。

The quality of the inferred specifications is critical to their suc-cessful use.In fact, most research on specification inference focuses on improving the quality of the inferred specifications.Yet, the rela-tionship between the principles employed by inference techniques, and the quality of the models they produce is not well understood, and existing studies have not pursued to improve this understanding in the context of real-world software.The goal of this paper is to systematically evaluate the principles behind existing inference techniques, particularly with respect to the input data used by the techniques, to understand the effects of these principles on the in-ferred models' quality and on the scalability of the inference.The main contributions of this work are (1) an enhanced understanding of how to best design and improve inference techniques, and (2) a novel technique that implements these improvements.

推断规格的质量对其成功使用至关重要。事实上，大多数关于规范推断的研究都集中在提高推断规范的质量上。然而，推理技术所采用的原理和它们所产生的模型的质量之间的关系还没有被很好地理解，现有的研究也没有在现实世界软件的环境中寻求提高这种理解。本文的目的是系统地评估现有推理技术背后的原理，特别是关于这些技术使用的输入数据，以理解这些原理对优选模型的质量和推理的可伸缩性的影响。这项工作的主要贡献是(1)增强了对如何最好地设计和改进推理技术的理解，以及(2)实现这些改进的新技术。

Existing mining techniques either (1) infer finite state machine (FSM) models that support the observed invocation sequences (also referred to as execution traces) [2,4,10,21,39,43,49,56], or (2) iden-tify high-level properties — declarative class and method invariants — by observing how a library's state (its internal variables) changes at runtime [11, 20, 57].Hence, the utility of the dynamic inference techniques in general, and FSM-inference techniques in particular, critically depends on how rich the inferred models are and how close they are to the true model of the system's behavior.The primary focus of FSM-inference research has been to improve the inferred models' precision (e.g., [10,39,41,49,56]).Recent research suggests that the inferred models' quality can be improved by extending the FSM inference that relies solely on execution traces, by also using internal state information available during execution [16, 41].In the limit, the Contractor algorithm demonstrates that FSM inference can rely exclusively on internal state information [18].

现有的挖掘技术或者(1)推断支持观察到的调用序列(也称为执行轨迹)的有限状态机模型([2，4，10，21，39，43，49，56)，或者(2)通过观察库的状态(其内部变量)在运行时[11，20，57 如何变化来识别高级属性(声明类和方法不变量)。因此，一般来说，动态推理技术的效用，特别是有限状态机推理技术的效用，关键取决于所推断的模型有多丰富，以及它们与系统行为的真实模型有多接近。有限状态机推理研究的主要焦点是提高推理模型的精度(例如，[10，39，41，49，56])。最近的研究表明，通过扩展仅依赖于执行轨迹的有限状态机推断，也通过使用执行期间可用的内部状态信息，可以提高推

断模型的质量[16，41]。在极限情况下，承包商算法证明有限状态机的推断可以完全依赖于内部状态信息[18]。

While existing evaluations of model inference research have made substantial contributions, they also have had significant limitations our work addresses.Most importantly, to properly compare inference techniques, it is necessary to understand how well they perform on real software, and how their underlying principles (e.g., reliance on inferred state invariants vs. reliance on execution traces) affect model quality.Many existing studies of model quality rely on simulated execution traces, as opposed to traces from actual software systems [10, 36, 39].As simulations are typically more controlled than real software, this runs the risk of inaccurately esti-

虽然现有的模型推理研究的评估已经做出了实质性的贡献，但是它们也有我们的工作所解决的重大局限性。最重要的是，为了恰当地比较引用技术，有必要了解它们在真实软件上的性能，以及它们的基本原理(例如，依赖推断的状态不变量与依赖执行轨迹)如何影响模型质量。许多现有的模型质量研究依赖于模拟的执行轨迹，而不是来自实际软件系统的轨迹[10，36，39]。由于模拟通常比真实的软件更受控制，这就有估计不准确的风险



178

mating how the techniques perform in the wild.Meanwhile, those studies performed on real software typically estimate the quality of the inferred models via proxies.For example, the quality is some-times measured via test coverage [41], the number of automatically detected faults [16, 21], or case-study analysis of manually discov-ered faults [3, 4, 18], instead of the more generalizable information retrieval metrics of precision and recall [36].

交配技术在野外的表现。同时，那些在真实软件上进行的研究通常通过代理来估计推断模型的质量。例如，质量有时是通过测试覆盖率[41]，自动检测故障的数量[16，21]，或人工发现故障的案例研究分析[3，4，18]来测量的，而不是精度和召回[36]的更一般化的信息检索度量。

Further, there are no studies to date that thoroughly measure the effects of the types of input information used by inference techniques on the quality of the resulting models.In particular, there are four possible strategies to dynamic FSM inference: 1.Traces-only: infer models from execution traces only.2.Invariants-only: infer models from invariants.3.Invariant-enhanced-traces: infer models from execution traces

此外，到目前为止，还没有研究能够全面衡量推理技术所使用的输入信息类型对最终模型质量的影响。特别地，有四种可能的策略来动态地进行有限状态机推断:1 .仅跟踪:仅从执行跟踪推断模型。2.仅不变量:从不变量推断模型。3.不变增强跟踪:从执行跟踪推断模型

and then enhance them with invariants.

然后用不变量增强它们。

4.Trace-enhanced-invariants: infer models from invariants and then enhance them with execution traces.

4.跟踪增强不变量:从不变量中推断模型，然后用执行跟踪增强它们。

In this context, the research questions "How do these strategies impact the quality of the inferred models?"and "Under what cir-cumstances are they effective?"remain unanswered.(Notably, the

empirical study conducted by Lo et al. [40] provides an initial com-parison of the first and the third strategies above.)

在这种背景下，研究的问题是“这些策略如何影响推断模型的质量？”“在什么情况下它们有效？”仍然没有答案。(值得注意的是，罗等人[40 进行的实证研究提供了上述第一和第三种策略的初步比较。)

To answer these questions, this paper provides an empirical study that compares the quality of models inferred by four techniques that, respectively, implement the above four strategies.The k-tail [5] inference algorithm, representing the traces-only strategy, is employed and enhanced by many model-inference techniques (e.g., [2, 3, 39, 41, 43, 49]).We select k-tail because k-tail mod-els turn out to already be highly precise in our evaluations, and further precision improvements would not affect our findings.CON-TRACTOR++, representing the invariants-only strategy, is our own extension of the Contractor technique [18] that creates models based solely on the inferred state invariants.CONTRACTOR++ addresses the limitations of Contractor when applied to dynamically inferred, as opposed to manually specified, state invariants.SEKT, represent-ing the invariants-enhanced-traces strategy, is our own extension of k-tail that uses state invariants to restrict the way k-tail merges execution traces.SEKT expands on Lorenzoli et al.'s gk-tail algo-rithm [41] by relaxing restrictions on how and which invariants are inferred for small subsets of the execution traces.Finally, TEMI, representing the trace-enhanced-invariants strategy, is a novel tech-nique we have developed as part of this research that first infers a model based on the state invariants [20], and then refines that model with the information from the execution traces.

为了回答这些问题，本文提供了一个实证研究，比较了分别实现上述四种策略的四种技术所推断的模型的质量。许多模型推理技术(例如，[2、3、39、41、43、49)采用并增强了代表仅跟踪策略的 k 尾[推理算法。我们选择 k-tail 是因为 k-tail 模型在我们的评估中已经非常精确，进一步的精度改进不会影响我们的发现。代表仅不变量策略的 CON-TRACTOR++，是我们自己对承包商技术[18]的扩展，它仅基于推断的状态不变量创建模型。承包商++解决了承包商在应用于动态推断(而不是手动指定)状态不变量时的局限性。SEKT 代表不变量增强跟踪策略，是我们自己对 k-tail 的扩展，它使用状态不变量来限制 k-tail 合并执行跟踪的方式。SEKT 扩展了 Lorenzoli 等人的 gk-tail 算法[41]，放宽了对执行轨迹的小子集如何和哪些不变量被推断的限制。最后，代表轨迹增强不变量策略的 TEMI 是我们作为本研究的一部分开发的一种新技术，它首先基于状态不变量[20]推导模型，然后利用来自执行轨迹的信息来改进该模型。

A systematic, careful evaluation of the four strategies has two crit-ical requirements.First, the findings must be generalizable.Second, they must be applicable to real-world systems.Our evaluation relies on nine widely-used, open-source libraries selected from a range of domains.Further, our evaluation derives the execution traces by running eight real-world, publicly-available applications that use the selected libraries.By doing so, our evaluation reflects more closely than most existing studies the scenario of an engineer encountering a poorly documented library and trying to infer a usage model from other open-source software uses of that library.We compare the quality — represented by precision and recall — of the models produced by the four strategies.We also assess two important and often overlooked aspects of model inference: the scalability of the strategies and their robustness to noise in the inputs.

对这四种策略进行系统、仔细的评估有两个关键要求。首先，这些发现必须具有普遍性。其次，它们必须适用于现实世界的系统。我们的评估依赖于从一系列领域中选择的九个广泛使用的开源库。此外，我们的评估通过运行使用所选库的八个现实世界中公开可用的应用程序来导出执行跟踪。通过这样做，我们的评估比大多数现有研究更接近地反映了一个工程师遇到一个文档不足的库并试图从该库的其他开源软件使用中推断出一个使用模型的场景。我们比较了由四种策略产生的模型的质量——以精确度和召回率为代表。我们还评估了模型推理的两个重要且经常被忽略的方面:策略的可伸缩性和它们对输入中噪声的鲁棒性。

We make four evidence-supported conclusions: 1.Invariants-only and trace-enhanced-invariants strategies produce

我们得出四个有证据支持的结论:1 .仅不变量和跟踪增强不变量策略产生

significantly higher recall than the traces-only and invariant-enhanced-traces strategies, while maintaining (or, in rare cases, minimally reducing) the already high precision.

显著高于仅跟踪和不变增强跟踪策略的召回率，同时保持(或者在极少数情况下，最小程度地降低)已经很高的精度。

2.In the general case, invariant-enhanced-traces and trace-enhanced-invariants strategies that combine the two types of execution in-formation slightly improve the precision of the inferred models, while maintaining the recall.

2.在一般情况下，结合两种执行信息的不变增强跟踪和跟踪增强不变策略略微提高了推断模型的精度，同时保持了召回率。

3.Invariant filtering that keeps only a limited set of relevant invari-ant types is crucial to enhancing the scalability of techniques that implement invariants-only and trace-enhanced-invariants strategies.

3.只保留有限一组相关不变量类型的不变过滤对于增强实现仅不变量和跟踪增强不变量策略的技术的可伸缩性至关重要。

4.While the quality of models inferred by invariants-only and trace-enhanced-invariants strategies is similar in most cases, com-bining an FSM inferred from invariants with execution traces circumvents the risks associated with noisy invariants.Such noise significantly reduces the quality of invariants-only strategy models, making it also perform worse than the traces-only and invariant-enhanced-traces strategies.

4.虽然在大多数情况下，由仅不变量和跟踪增强不变量策略推断出的模型质量相似，但是将从不变量推断出的有限状态机与执行跟踪相结合可以规避与噪声不变量相关的风险。这种噪声显著降低了仅不变量策略模型的质量，使得它的性能也比仅跟踪策略和不变量增强跟踪策略差。

The remainder of the paper is organized as follows.Section 2 overviews the background.Section 3 details the techniques repre-senting the four strategies.Section 4 evaluates those techniques, while Section 5 discusses the impact of the results.Section 6 places our work in the context of related research.Finally, Section 7 sum-marizes the paper's contributions.

本文的其余部分组织如下。第 2 节概述了背景。第 3 节详细介绍了代表四种策略的技术。第 4 节评估这些技术，第 5 节讨论结果的影响。第 6 节将我们的工作放在相关研究的背景下。最后，第七部分总结了论文的贡献。

## 2.BACKGROUND

## 2.背景

This section provides the background necessary for the discus-sions in this paper.Section 2.1 introduces StackAr, our running example data structure.Section 2.2 defines the MTS formalism used by three of the model-inference techniques in Section 3 and discusses how execution traces map to an MTS. Finally, Section 2.3 defines program state and discusses state invariants.

本节为本文的讨论提供了必要的背景。第 2.1 节介绍了 StackAr，我们的运行示例数据结构。第 2.2 节定义了第 3 节中三种模型推理技术使用的中期战略形式，并讨论了执行跟踪如何映射到中期战略。最后，第 2.3 节定义了程序状态并讨论了状态不变量。

2.1 Example Library: StackAr

2.1 示例库:StackAr

StackAr is a Java implementation of a stack, distributed with Daikon [15, 20].Initialized with an integer capacity, StackAr has six public methods: push(x), top(), topAndPop(), makeEmpty(), isEmpty(), and isFull().Internally, StackAr represents its stack as an array (theArray) and has a pointer to the top of the stack (topOfStack).A push() on a full stack generates an exception.A top() or topAndPop() on an empty stack returns null.

StackAr 是一个堆栈的 Java 实现，与 Daikon [15，20]一起分发。StackAr 以整数容量初始化，有六种公共方法:push(x)、top()、topAndPop()、makeEmpty()、isEmpty()和 isFull()。在内部，StackAr 将其堆栈表示为数组(ArArray)，并有一个指向堆栈顶部的指针(topOfStack)。完整堆栈上的推送()会生成异常。空堆栈上的 top()或 topAndPop()返回 null。

2.2 Modal Transition System (MTS)

2.2 模式转换系统

An MTS [33] is an FSM-based model with labeled transitions between states.A state represents a specific point in the execution of a system or module;a transition represents the system's change from one state to another, caused by some invocation.In an MTS, there are two explicit kinds of transitions: required, which are transitions that are certain to occur and are common to all FSMs, and maybe, which are uncertain transitions unique to MTSs.We use the notation s

MTS [33]是一种基于有限状态机的模型，带有标记的状态间转换。状态表示系统或模块执行中的特定点;转换表示系统从一种状态到另一种状态的变化，这种变化是由一些调用引起的。在多边贸易体制中，有两种明确的过渡类型:必需的，即肯定会发生并为所有金融服务机构所共有的过渡，也可能是多边贸易体制特有的不确定过渡。我们使用符号 s

l

l

$-\rightarrow_r s$ and $s\ l -\rightarrow_m s$ , respectively, to denote required and maybe l-labeled transitions between states s and s .

$\rightarrow_r s$ 和 $s\ l \rightarrow_m s$，分别表示状态 s 和 s 之间所需的和可能 l 标记的转换。

The most common input to a model-inference algorithm is a set of observed execution traces.An execution trace — a runtime record-ing of public method invocations and internal data values between those invocations — can be represented by an MTS with states cor-responding to variable values and transition labels composed of the method name, input values, and return value.

模型推理算法最常见的输入是一组观察到的执行轨迹。执行跟踪——公共方法调用和这些调用之间的内部数据值的运行时记录——可以用状态对应于变量值和由方法名、输入值和返回值组成的转换标签的 MTS 来表示。

2.3 Program State and State Invariants

2.3 程序状态和状态不变量

A program's concrete state is represented by the values of the program's variables at a given snapshot in the program's execution.However, for non-trivial programs, there exist intractably many concrete states.For example, for StackAr, there are an infinite

程序的具体状态由程序执行过程中给定快照上的程序变量值来表示。然而，对于不平凡的程序，有许多难以解决的具体状态。例如，对于 StackAr，有一个无限



179

DataStructures.StackAr:::CLASS this.topOfStack >= -1

数据结构。StackAr:::将此分类。拓扑结构> = -1

this.topOfStack <= size(this.theArray[])-1

这个。拓扑结构< =大小(这个。射线[])-1

DataStructures.StackAr.push(java.lang.Object):::ENTER this.topOfStack < size(this.theArray[..])-1 this.topOfStack >= -1

数据结构。*输入这个..])-1 此。拓扑结构> = -1

DataStructures.StackAr.push(java.lang.Object):::EXIT103 orig(this.topOfStack) < size(this.theArray[..])-1 this.topOfStack >= 0

数据结构。* exit 103 origin(this . topofstack)< size(this . Array[..])-1 此。拓扑结构> = 0

this.topOfStack - orig(this.topOfStack) - 1 == 0 size(this.theArray[..]) == orig(size(this.theArray[..]))

这个。拓扑结构-原点(这个。拓扑结构)- 1 == 0 大小(这个。射线[..]) ==原始(大小(这个是[]..]))

Figure 1: A subset of Daikon's program invariants on StackAr.

图 StackAr 上 Daikon 程序不变量的子集。

number of concrete states represented by the contents of theArray.Therefore, it is common to consider and reason about abstract pro-gram states [17,62] defined with first-order predicates over program variable values: Different program states correspond to different combinations of predicate evaluations.For StackAr, one reasonable predicate that can be used to define abstract state is "is the stack not full?"(topOfStack < size(theArray)−1).

由数组内容表示的具体状态数。因此，通常考虑和推理抽象程序状态[17，62]定义为程序变量值上的一阶谓词:不同的程序状态对应于谓词评估的不同组合。对于 StackAr 来说，一个可以用来定义抽象状态的合理谓词是“堆栈没有满吗？”(拓扑结构<尺寸(半径)1)。

Program-state invariants can be used to automatically extract relevant predicates.Invariants hold true at certain program execution points.For example, an object-level invariant, such as size ≥ 0 holds at all program points.While developers can manually specify program invariants in the code or in other documentation [18, 46], static and dynamic analyses can automatically infer invariants.

程序状态不变量可以用来自动提取相关谓词。不变量在某些程序执行点成立。例如，对象级不变量(如大小≥ 0)适用于所有程序点。虽然开发人员可以在代码或其他文档[18，46 中手动指定程序不变量，但是静态和动态分析可以自动推断不变量。

A commonly used tool for automatic invariant inference is Dai-kon [20], which observes data values of program executions and infers invariants that hold over all observed executions.The in-ferred invariants consist of method pre- and postconditions and object invariants.Figure 1 shows eight invariants Daikon infers for StackAr: two object-level invariants, and two preconditions and four postconditions for push().As preconditions and object invariants help determine which methods can execute in a particular state, the predicates that appear in Daikon-inferred preconditions and object invariants are good candidates for defining abstract pro-gram state.For StackAr, Daikon reports four predicates: P1 = (topOfStack ≥ −1), P2 = (topOfStack ≥ 0),

自动不变量推理的常用工具是戴孔[20]，它观察程序执行的数据值，并推断所有观察到的执行的不变量。优选的不变量包括方法前后条件和对象不变量。图 1 显示了 Daikon 为 StackAr 推断的八个不变量:两个对象级不变量，两个推送的前提条件和四个后条件()。因为先决条件和对象不变量有助于确定哪些方法可以在特定状态下执行，所以出现在 Daikon 推断的先决条件和对象不变量中的谓词是定义抽象程序状态的良好候选。对于 StackAr，Daikon 报告了四个谓词:P1 =(拓扑结构≥1)，P2 =(拓扑结构≥ 0)，

P3 = (topOfStack < size(theArray)−1), and P4 = (topOfStack ≤ size(theArray)−1).

P3 =(拓扑结构<尺寸(半径)1)，P4 =(拓扑结构≤尺寸(半径)1)。

Four boolean predicates can define 2 = 16 abstract program states.However, many of these states can be automatically eliminated be-cause of predicate interdependencies.For example, StackAr's P2 cannot be true when P1 is false.

四个布尔谓词可以定义 2 = 16 个抽象程序状态。然而，由于谓词相互依赖，这些状态中的许多可以被自动消除。例如，当 P1 是假的时候，斯塔克的 P2 不可能是真的。

KTAIL(trace-set traces, int k) 1 MTS T = PTAl(traces) 2 for each pair T.s1,T.s2 in T.S 3 if TAIL(T.s1, k) = TAIL(T.s1, k) 4 MERGE(T.s1, T.s1) 5 return T

如果 TAIL(T.s1，k) = TAIL(T.s1，k) 4 MERGE(T.s1，T.s1) 5 返回 T，则 KTAIL(迹线集迹线，int k) 1 MTS T = PTAl(迹线)2

Figure 2: The k-tail algorithm.

图 2:k 尾算法。

## 3.INFERENCE ALGORITHMS

## 3.推理算法

This section details the four algorithms that respectively imple-ment each the four model inference strategies (recall Section 1).k-tail (Section 3.1) implements the traces-only strategy by reason-ing exclusively about the invocation sequences observed in the execution traces.Contractor [18]

(Section 3.2) builds a model by reasoning about the potentially allowed invocation sequences based on manually specified state invariants.To apply Contractor on dy-namically inferred invariants, we have developed CONTRACTOR++ via a set of non-trivial extensions of Contractor.SEKT (Section 3.3) is our extension of k-tail inference that implements the invariant-enhanced-traces strategy by considering the program states extracted from invariants as a k-tail merging criterion.Finally, TEMI (Sec-tion 3.4) implements the trace-enhanced-invariants strategy by first building an MTS that adheres to the invariants and then refining it according to the execution traces.

本节详细介绍了分别实现四种模型推理策略的四种算法(回忆第 1 节)。k-tail(第 3.1 节)通过专门推理执行跟踪中观察到的调用序列来实现仅跟踪策略。承包商[18](第 3.2 节)基于手动指定的状态不变量，通过推理潜在允许的调用序列来构建模型。为了将承包商应用于动态推断的不变量，我们通过承包商的一系列重要扩展开发了承包商++程序。SEKT(第 3.3 节)是我们对 k 尾推理的扩展，它通过考虑从不变量中提取的程序状态作为 k 尾合并标准来实现不变增强跟踪策略。最后，TEMI(第 3.4 节)实现了跟踪增强不变量策略，首先构建一个遵循不变量的 MTS，然后根据执行跟踪对其进行细化。

### 3.1 Traces-Only: Traditional k-tail

### 3.1 仅跟踪:传统 k 形尾翼

The existing k-tail algorithm [5] concisely captures a library's API protocol as an MTS with required-only transitions by merging the states from the execution traces.The algorithm merges every pair of states with identical sequences of the next k invocations (hence "k-tail").Figure 2 details the algorithm.In the first step (line 1), k-tail builds a prefix-tree-acceptor (PTA).The PTA is an MTS obtained by merging the initial states of each trace into a single initial state, and from that state, merging all those states that are reached by the same invocation sequence (e.g., states s1 through s4 of Trace 1, are merged with states s1 through s4 of Trace 4, respectively).Subsequently, lines 2-4 analyzes each pair of states in the PTA for tail equivalence;line 4 merges the equivalent states.

现有的 k-tail 算法[5]通过合并来自执行轨迹的状态，简洁地将库的应用编程接口协议捕获为只需要转换的 MTS。该算法将每对状态与接下来 k 次调用的相同序列合并(因此称为“k 尾”)。图 2 详细描述了算法。在第一步(第 1 行)，k-tail 构建前缀树接受者(PTA)。PTA 是通过将每个轨迹的初始状态合并成单个初始状态而获得的 MTS，并且从该状态，合并由相同调用序列到达的所有那些状态(例如，轨迹 1 的状态 s1 至 s4，分别与轨迹 4 的状态 s1 至 s4 合并)。随后，第 2-4 行分析 PTA 中每对状态的尾部等价；第 4 行合并了等价的状态。

The k-tail algorithm relies on selecting an appropriate k. This choice typically involves a tradeoff between precision (smaller k im-plies more spurious merges due to the limited scope) and complete-ness/recall (larger k implies fewer merges and less generalization) of the generated model.Because of that, the existing k-tail-based techniques (e.g., [39,41,49]) have aimed to improve the algorithm's precision with smaller k. Despite some improvements, the existing k-tail-based algorithms still suffer from limitations described next.

k-tail 算法依赖于选择合适的 k。这种选择通常需要在生成模型的精度(较小的 k-im 由于范围有限而带来更多的虚假合并)和完全性/召回(较大的 k 意味着较少的合并和较少的泛化)之间进行权衡。因此，现有的基于 k-tail 的技术(例如，[39，41，49])旨在用较小的 k 来提高算法的精度。尽管有一些改进，但是现有的基于 k-tail 的算法仍然受到下面描述的限制。

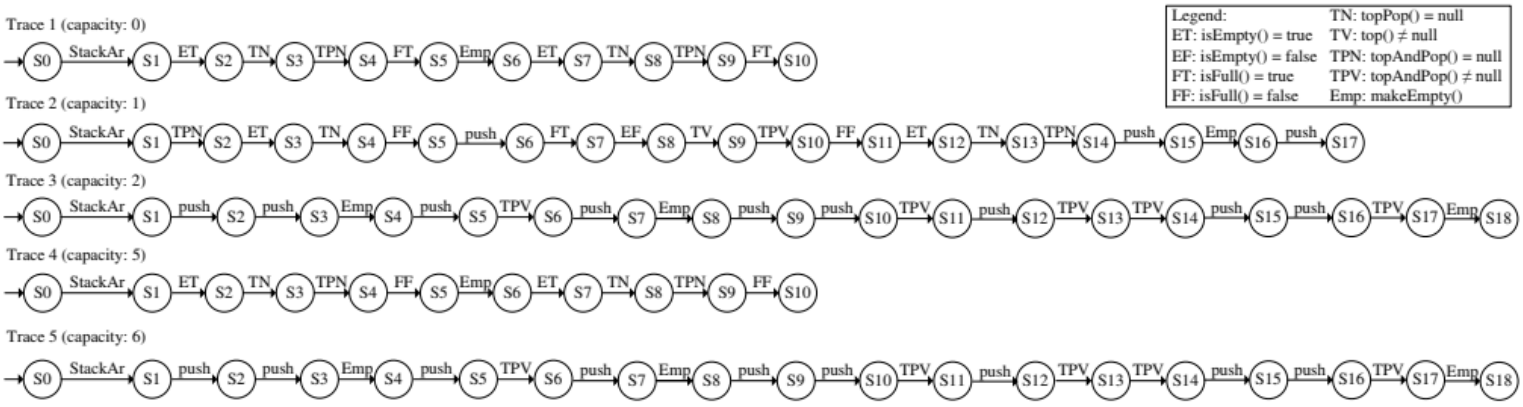


Figure 3: Five example StackAr invocation traces.

图 3:五个 StackAr 调用跟踪示例。



180

CONTRACTOR(inv-set invariants) 1 MTS T, T.S = 0/

承包商(库存集不变量)1 公吨，公吨= 0/

2 for each tuple enabled ∈ invariants.methods 3 clause statePred = 0/ 4 for each method ∈ enabled

每个元组启用 2∈不变量。方法 3 子句状态对于每个方法∈启用 Pred = 0/ 4

5 statePred = statePred∧method.preCond 6 for each method ∈/ enabled

5 状态预解码=状态预解码 Σ 方法。每个方法预解码 6∑/使能

7 statePred = statePred∧ method.preCond 8 if SMT-ISCONSISTENT(statePred) 9 add T.senabled to T.S

7 状态预定义=状态预定义∪方法。如果 SMT-ISCONSISTENT(状态预定义)9 将启用的测试添加到测试系统，则预定义 8

10 for each pair T.s1,T.s2 in T.S and method in inv.methods 11 if SMT-ISCONSISTENT(T.s1 ∧method.preCond) 12 and SMT-ISCONSISTENT(T.s2 ∧method.postCond) 13 add T.s1 method −→r T.s2 to T.transitions 14 return T

10 对于每对 T.s1、T.s 中的 T.s2 和 inv 中的方法。方法 11 如果 SMT-ISCSINSTENT(T.s1∧方法.预处理)12 和 SMT-ISCSINSTENT(T . S2∧方法.后处理)13 添加 T . S1 方法→r T.s2 到 T .转换 14 返回 T

Figure 4: The Contractor algorithm.

图 4:承包商算法。

To illustrate k-tail and its shortcomings, we use five StackAr invocation traces, corresponding to creating and using stacks of different capacities (Figure 3).Let us consider how 2-tail that considers

the method return values as part of its merging criterion works on these traces.The algorithm correctly merges states s1 and s6 in Trace 1 because the two following invocations from each state are isEmpty()=true and top()=null.The algorithm also merges states s1 in Trace 1 and s11 in Trace 2.However, this merge is imprecise as it allows a non-zero-capacity stack to change capacity to zero after an invocation of isFull() from s10 to s11 in Trace 2.The k-tail models are also incomplete for those traces in which pairs of methods happen to be, but are not actually required to be, invoked in a specific order (e.g., top() is always invoked after isEmpty() in Figure 3).The existing k-tail-based techniques (e.g., [39, 41, 49]) all suffer from these limitations.

为了说明 k-tail 及其缺点，我们使用了五个 StackAr 调用跟踪，对应于创建和使用不同容量的堆栈(图 3)。让我们考虑将方法返回值作为合并标准的一部分的双尾如何在这些跟踪上工作。该算法正确地合并了跟踪 1 中的状态 s1 和 s6，因为来自每个状态的以下两个调用是 isEmpty()=true 和 top()=null。该算法还合并了轨迹 1 中的状态 s1 和轨迹 2 中的状态 s11。但是，这种合并是不精确的，因为它允许非零容量堆栈在跟踪 2 中从 s10 到 s11 调用 isFull()后将容量更改为零。k-tail 模型对于那些跟踪来说也是不完整的，在这些跟踪中，方法对碰巧以特定的顺序被调用，但实际上并不需要被调用(例如，在图 3 中，top()总是在 isEmpty()之后被调用)。现有的基于 k-tail 的技术(例如，[39、41、49])都受到这些限制。

3.2 Invariants-Only: CONTRACTOR++

3.2 不变量-仅:承包商++

As mentioned earlier, CONTRACTOR++ comprises two parts: (1) the core algorithm Contractor [18], a recent algorithm that creates MTS models exclusively based on program invariants, and (2) our enhancements to Contractor take enable it to work on inferred, as opposed to manually specified, program invariants.

如前所述，承包商++包括两个部分:(1)承包商[18 的核心算法，这是一种最近专门基于程序不变量创建 MTS 模型的算法，以及(2)我们对承包商的增强使其能够处理推断的程序不变量，而不是手动指定的程序不变量。

Figure 4 lists the Contractor algorithm.Contractor uniquely char-acterizes the model's states by the combination of methods enabled in each state (lines 2-9 build the state set T.S).This abstraction is thus referred to as enabledness.A state (i.e., a combination of enabled methods) is legal if the preconditions of the enabled methods are consistent with one another.The algorithm checks for consistency of the preconditions in lines 2-9 with the help of an off-the-shelf Satisfiability Modulo Theories (SMT) solver, such as Yices [61].Lines 10-14 create a transition on a method between two states if that method's precondition is satisfied in the source state and the postcondition is satisfied in the target state.

图 4 列出了承包商算法。承包商通过在每个状态中启用的方法的组合来唯一表征模型的状态(第 2-9 行构建状态集)。这种抽象因此被称为使能性。如果启用方法的前提条件彼此一致，则状态(即启用方法的组合)是合法的。该算法借助现成的可满足性模理论(SMT)求解器，如伊奇·[61]，检查第 2-9 行中先决条件的一致性。如果方法的前提条件在源状态中得到满足，而后条件在目标状态中得到满足，则第 10-14 行在该方法上创建两种状态之间的转换。

To use Contractor in our evaluation, we had to enhance its inputs in two significant ways, without which both the accuracy and scal-ability of the algorithm would be notably lower.We refer to these enhancements as CONTRACTOR++, and use CONTRACTOR++ as the implementation of the invariants-only inference strategy.

为了在我们的评估中使用承包商，我们必须以两种重要的方式增强其输入，没有这两种方式，算法的准确性和可伸缩性都会明显降低。我们称这些增强为 CONTRACTOR++,并使用 CONTRACTOR++作为仅不变量推理策略的实现。

We refer to the first enhancement as Method Distinction: Since Contractor does not consider predicates of the methods' output values, in CONTRACTOR++ we represent each (method, return-value) combination as a distinct method with its own invariants.Otherwise, for example, the model inferred for StackAr would have only two states: the first one with all methods enabled, and the second one with push() prohibited and all other methods allowed.Such a model is imprecise as it allows, for example, pushing on a stack of 0-capacity, or popping non-null values from an empty stack.

我们将第一个增强称为方法区分:因为承包商不考虑方法输出值的谓词，所以在承包商++中，我们将每个(方法、返回值)组合表示为具有自己不变量的不同方法。否则，例如，为 StackAr 推断的模型只有两种状态:第一种是启用所有方法的状态，第二种是禁止 push()和允许所有其他方法的状态。这种模型是不精确的，因为它允许，例如，推动 0 容量的堆栈，或者从空堆栈弹出非空值。

ADVANCEDMERGE(st T1.S, st T2.P, int k, set pred) 1 if Tail(T1.S, k) 6= Tail(T2.P, k) 2 return false

高级合并(st T1。圣 T2。p，int k，设置 pred) 1 如果尾部(T1。s，k) 6=尾巴(T2。p，k) 2 返回 false

3 if EVALGLOBAL(T1.S, pred) 6= EVALGLOBAL(T2.P, pred) 4 return false 5 return true

3 if EVALOGLE(T1。s，pred) 6= EVALGLOBAL(T2)。p，pred) 4 返回假 5 返回真

Figure 5: The SEKT algorithm uses ADVANCEDMERGE to de-termine if two states should be merged.

图 SEKT 算法使用 ADVANCEDMERGE 来确定两个状态是否应该合并。

We refer to the second enhancement as Invariant Filtering: In-stead of using manually-specified invariants, CONTRACTOR++ uses Daikon-inferred invariants, filtered to avoid less meaningful invari-ants and to make the algorithm scale, since dynamically inferred invariants typically have higher complexity than the manually spec-ified ones [46].We consider the relational invariants on boolean and integer variables (e.g., IntEqual, IntGreaterThan), and the IsNull invariant on objects.Further, we consider internal variables up to a depth of one (i.e., we consider an object's fields, but not those fields' fields).For collections, we consider their sizes but not their elements.Without these enhancements, the original Contractor algorithm's models are of significantly lower quality than reported below for CONTRACTOR++.Note that the same set of filters are employed in (and were, in fact, originally developed for) TEMI.

我们将第二个增强称为不变量过滤:承包商++使用戴康推断的不变量，而不是使用手动指定的不变量，过滤是为了避免不太有意义的不变量并使算法缩放，因为动态推断的不变量通常比手动指定的不变量具有更高的复杂性，[46]。我们考虑布尔变量和整数变量(例如整数、整数)上的关系不变量，以及对象上的 IsNull 不变量。此外，我们考虑深度为 1 的内部变量(即，我们考虑对象的字段，但不考虑那些字段的字段)。对于收藏品，我们考虑的是它们的大小，而不是它们的元素。如果没有这些增强，原始承包商算法的模型的质量将明显低于下面针对承包商++请注意，TEMI 采用了相同的过滤器组(事实上，最初是为该国开发的)。

3.3 Invariant-Enhanced-Traces: SEKT

3.3 不变-增强-轨迹:SEKT

While k-tail has been used extensively in prior work (recall Sec-tion 3.1), our State-Enhanced k-tail, SEKT, is the first algorithm that extends k-tail using program state information inferred from the full set of observed executions.The most closely related algorithm, proposed by Lorenzoli et al. [41], infers invariants only for the lim-ited set of states that are merged during PTA construction (recall Figure 2).Due to this limited scope for learning relevant invariants, Lorenzoli et al.'s algorithm may err both in allowing and rejecting merges, which may decrease both the precision and the recall of the resulting model [40].For example, this approach merges the states in StackAr's Trace 1 and Trace 4 from Figure 3 because they follow the same invocation sequence, despite the different method parameter and return values.Consequently, the resulting merged trace erroneously implies that a stack of size 0 is the same as one of size 5 even though isFull returns different values.

虽然 k-tail 已经在以前的工作中广泛使用(回忆第 3.1 节)，但是我们的状态增强 k-tail，SEKT，是第一个使用从全部观察到的执行中推断出的程序状态信息来扩展 k-tail 的算法。由洛伦佐利等人[41 提出的最密切相关的算法，仅推断在 PTA 构建期间合并的有限状态集的不变量(回想图 2)。由于学习相关不变量的范围有限，洛伦佐利等人的算法在允许和拒绝合并时都可能出错，这可能降低最终模型[40]的精度和召回率。例如，这种方法合并了图 3 中 StackAr 的跟踪 1 和跟踪 4 中的状态，因为它们遵循相同的调用序列，尽管方法参数和返回值不同。因此，得到的合并跟踪错误地暗示大小为 0 的堆栈与大小为 5 的堆栈相同，即使 isFull 返回不同的值。

In contrast to Lorenzoli et al.'s algorithm, SEKT modifies the k-tail algorithm by adding a new global merge requirement: The merging states must correspond to the same abstract program state.Method ADVANCEDMERGE in Figure 5 details this merging crite-rion and is incorporated into k-tail by replacing the original con-dition specified in line 3 of Figure 2.The pred parameter of AD-VANCEDMERGE is the set of predicates that define the program states, and T1.S and T2.P are the two potential to-be-merged states.Lines 1-2 of ADVANCEDMERGE test the tail similarity;lines 3-4 check for matching program states.The state matching compares whether the two concrete variable evaluations correspond to the same abstract program state (EVALGLOBAL).

与 Lorenzoli 等人的算法不同，SEKT 通过添加新的全局合并要求来修改 k-tail 算法:合并状态必须对应于相同的抽象程序状态。图 5 中的方法 ADVANCEDMERGE 详细说明了这种合并标准，并通过替换图 2 第 3 行中指定的原始条件将其合并到 k 尾中。AD-VANCEDMERGE 的 pred 参数是定义程序状态的谓词集，T1。美国和 T2。p 是两个可能合并的状态。ADVANCEDMERGE 的第 1-2 行测试尾部相似性；第 3-4 行检查匹配的程序状态。状态匹配比较两个具体变量评估是否对应于相同的抽象程序状态(EVALGLOBAL)。

The new merging condition can prevent erroneous merges.For example, SEKT avoids the spurious merge of s1 in Trace 1 and s11 in Trace 2 in the StackAr traces from Figure 3 (recall Section 3.1): The condition in line 3 of ADVANCEDMERGE is satisfied, thus rejecting the merge, since only predicates P1 and P4 (from Section 2.3) are true in s1, but only P1 and P3 are true in s11.

新的合并条件可以防止错误合并。例如，SEKT 避免了图 3 中堆栈跟踪的跟踪 1 中 s1 和跟踪 2 中 s11 的虚假合并(回忆第 3.1 节):ADVANCEDMERGE 第 3 行中的条件得到满足，因此拒绝合并，因为在 s1 中只有谓词 P1 和 P4(来自第 2.3 节)为真，但在 s11 中只有 P1 和 P3 为真。

3.4 Trace-Enhanced-Invariants: TEMI

3.4 跟踪增强不变量:TEMI

Trace-Enhanced MTS Inference, TEMI, infers an MTS that in-cludes but differentiates (1) the behavior asserted legal by the in-

追踪增强的多边贸易体系推断，TEMI，推断出一个多边贸易体系，它包括但区分(1)由 in-声称合法的行为



181

GENERATEINVARIANTMTS(set pred, inv-set invariants) 1 MTS invariantMTS, set toProcess,isProcessed = 0/ 2 set predCombinations = COMBINE(pred) 3 for each combination ∈ predCombinations 4 if SMT-ISCONSISTENT(combination) 5 add combination to invariantMTS.states 6 add invariantMTS.initSt to toProcess 7 while toProcess 6= 0/ 8 currentSt = toProcess.pop 9 add currentSt to isProcessed 10 for each methodInv ∈ invariants

GENEINVARIANTMS(设置预变量，inv-set 不变量)1 MTS 不变量 MTS，设置为进程，ISproceed = 0/2 设置预组合= COMBINE(预变量)3 用于每个组合∈预组合 4 如果 SMT-ISCONSIST(组合)5 将组合添加到不变量 MTS，状态 6 将不变量 MTS.initSt 添加到进程 7，而进程 6= 0/ 8 当前 t =进程. pop 9 将当前 t 添加到 isProcessed 10 用于每个方法∈

11 if SMT-ISCONSISTENT(methodInv.pre∧currentSt) 12 for each targetSt ∈ invariantMTS.states 13 if SMT-ISCONSISTENT(methodInv.post∧targetSt) 14 if targetSt ∈/ isProcessed add targetState to toProcess 15 add currentSt methodInv.name −→m targetSt to invariantMTS.transitions 16 return invariantMTS

11 如果每个目标的 SMT-ISCONSISTENT(method inv . pre∧Current St)12 状态 13 如果目标的 SMT-ISCONSISTENT(method inv . post∧Target St)14 如果目标已处理，将目标状态添加到进程 15 添加当前方法名称→m 目标到不变量。转换 16 返回不变量

Figure 7: Constructing of an invariant-based MTS.

图 7:构建一个基于不变量的多边贸易体系

variants and (2) the behavior observed in the traces.TEMI consists of two phases.The first phase, conceptually similar to CONTRAC-TOR++, constructs an MTS with only maybe transitions, capturing all invocation sequences of an object's interface allowed by the in-variants.We call this model an invariant-based MTS. The second phase promotes transitions observed in the traces from maybe to re-quired.The remaining maybe transitions stem from generalizations performed during invariant inference.As we demonstrate later in the paper, these generalizations are error-prone when working with a partial, limited set of execution traces.TEMI is loosely inspired by our earlier work on refining requirements-level use-case scenar-ios [29].We have previously outlined an early version of TEMI [31], but the algorithmic details and evaluation in this paper are new.

变体和(2)在轨迹中观察到的行为。TEMI 由两个阶段组成。第一阶段，在概念上类似于 CONTRAC-TOR++，构建一个只有可能转换的 MTS，捕获一个对象接口的所有调用序列，这些调用序列被 in 变体所允许。我们称这个模型为基于不变量的多边贸易体制。第二阶段促进了从可能到需要的跟踪中观察到的转变。其余的可能转换源于在不变推理过程中执行的归纳。正如我们在本文后面所展示的，当使用部分的、有限的一组执行跟踪时，这些概括容易出错。TEMI 从我们早先关于提炼需求级用例场景-ios [29]的工作中得到些许启发。我们之前已经概述了早期版本的 TEMI·[31]，但是本文中的算法细节和评估是新的。

Phase I: Synthesis of the Invariant-Based MTS

第一阶段:基于不变量的多边贸易体制的综合

The method GENERATEINITIALMTS (Figure 7) synthesizes an invariant-based MTS. It first constructs the prospective state space invariantMTS.states (lines 3-5) based on the set pred of predicates from method preconditions.For each possible combination of the predicate evaluations (line 2), GENERATEINVARIANTMTS uses Yices to check if that combination, in conjunction with object in-variants, is satisfiable (line 4).For StackAr, Daikon inferred 4 predicates (recall Section 2.3);Yices rejects as unsatisfiable every predicate combination with P1 = false and P2 = true.

方法 GENERATEINITIALMTS(图 7)合成了一个基于不变量的状态空间。它首先根据方法前提条件中谓词的集合 pred 构造预期的状态空间不变量状态(第 3-5 行)。对于谓词评估的每个可能的组合(第 2 行)，GENERATEINVARIANTMTS 使用 ices 来检查该组合以及对象内变量是否可满足(第 4 行)。对于 StackAr，Daikon 推断出 4 个谓词(回忆第 2.3 节)；易拒绝 P1 =假，P2 =真的每一个谓词组合。

After determining the valid states, GENERATEINVARIANTMTS creates transitions between those states (the loop in lines 7-15).Each transition added in line 15 has a source state that satisfies the appropriate method preconditions, and a destination state that satisfies the postconditions, similarly to CONTRACTOR++.The resulting invariant-based MTS contains a state for every reachable program state and a transition for every invocation sequence that is legal according to the invariants [29].

在确定有效状态后，GENERATEINVARIANTMTS 创建这些状态之间的转换(第 7-15 行中的循环)。第 15 行中添加的每个转换都有一个满足适当方法先决条件的源状态和一个满足后置条件的目标状态，类似于 CONTRACTOR++。得到的基于不变量的 MTS 包含每个可达程序状态的状态和根据不变量[29]合法的每个调用序列的转换。

Figure 8 depicts the invariant-based MTS for StackAr.Although the largest theoretical state space for StackAr is 2 = 16 states, with an additional initial state, the generated invariant-based MTS has only 5 states (including the initial state).There are several self-transitions that capture methods that do not change the program state.By contrast, the k-tail-based algorithms implicitly consider every method to be state-changing.

图 8 描述了 StackAr 基于不变量的 MTS。虽然 StackAr 最大的理论状态空间是 2 = 16 个状态，加上一个附加的初始状态，但生成的基于不变量的 MTS 只有 5 个状态(包括初始状态)。有几个自转换捕获不改变程序状态的方法。相比之下，基于 k-tail 的算法隐含地认为每种方法都是状态改变的。

TEMI's construction of the invariant-based MTS is conceptually similar to CONTRACTOR++ as it uses a predicate-based abstrac-tion of the program state.In contrast to Contractor, whose predi-cates are the full method preconditions, TEMI uses the individual clauses that appear in the invariants.The reason we choose this finer-grain abstraction is that the automatically inferred postcondi-

TEMI 构建的基于不变量的多边贸易体系在概念上类似于承包商++，因为它使用了基于谓词的程序状态抽象。与承包商相反，承包商的预测是完整的方法先决条件，TEMI 使用出现在不变量中的单独条款。我们选择这种细粒度抽象的原因是自动推断出的后置条件-

S0

S0

S4 S3

S4·S3

S1 S2 push?

S1·S2 推?

ET?, FT?, Emp?, TN?, TPN?

外星人？英国《金融时报》？，Emp？田纳西州？，主题方案网络?

ET?, FF?, Emp?, TN?, TPN?

外星人？，FF？，Emp？田纳西州？，主题方案网络?

push?, EF?, FF?, TV?, TPV?

推？，英孚？，FF？电视？，TPV?

EF?, FT?, TV?

英孚？英国《金融时报》？电视?

TPV?

TPV?

Emp?, TPV?

电磁脉冲。，TPV?

push?

推?

StackAr?

史黛卡?

StackAr?

史黛卡?

push?

推?

Emp?, TPV?

电磁脉冲。，TPV?

Figure 8: The invariant-based StackAr MTS.

图 8:基于不变量的堆栈结构

REFINEINVARIANTMTS(MTS invMTS, set traces) 1 currentSt = invMTS.initialState 2 for each currentEv ∈ traces 3 for each nextSt ∈ invMTS.states

REFINEVARIANTMS(MTS inv MTS，设置走线)1 电流测试= invMTS.initialState 2 用于每个电流测试 v ∈走线 3 用于每个下一个测试∈inv MTS . state

4 if SMT-ISCONSISTENT(nextSt∧currentEv.post)∧

4 如果 SMT-ISCONSISTENT(下一秒∧当前电压后)∨

currentSt currentEv −→m nextSt

电流 t 电流 v→m 下一步

5 REFINESTATE(currentSt,nextSt,currentEv,invMTS) 6 currentSt = nextSt

5 REFINESTATE(当前时间、下一时间、当前时间、当前时间)6 当前时间=下一时间

7 for each st1 ∈ invMTS.refinedSt 8 for each st1 l −→m st2 ∈ invMTS.transitions 9 if ∃st1 l −→r st3 remove st1 l −→m st2 10 for each state1,state2 ∈ invMTS.refinedSt 11 where state1.programSt = state2.programSt 12 if st1.outTrans ≈ st2.outTrans MERGE(st1, st2) 13 return invMT S

7 针对每个 st1 ∈ invMTS .针对每个 st1 l→m st2∈invMTS .转换 9 如果∃st1 l→r st3 移除 st1 l→m st2 10 针对每个状态 1，状态 2 ∈ invMTS.refinedSt 11 其中状态 1.programSt = state2.programSt 12 如果 st1 . out trans≈st2 . out trans merge(st1，st2) 13 返回 invm ts

REFINESTATE(state currentSt, state nextSt, event currentEv, MTS invMTS) 1 if currentSt = nextSt require currentStcurrentEv −→r nextSt 2 else

REFINESTATE(状态当前时间、状态下一时间、事件当前时间、中期战略投资)1，如果当前时间=下一时间需要当前时间→r 下一时间 2

3 add nextStto invMTS.states, rename nextSt to nextSt4 replace currentStcurrentEv −→γ nextStwith currentStcurrentEv −→r nextSt5 for each nextStl −→γotherSt in invMTS.transitions 6 if nextSt6= otherSt add nextStl −→γotherSt to invMTS.transitions 7 else add nextStl −→γnextStto invMTS.transitions

3 将 nextSt 添加到 invMTS.states，将 nextSt 重命名为 nextSt4，用 CurrentCurrentev→r next St 5 替换 invMTS 中的每个 next Stl→γother St。如果 nextSt6 = otherSt，则转换 6 将 next St→γother St 添加到 invMTS。转换 7 否则将 next STl→γnext St 添加到 invMTS 转换。

Figure 9: Refining the invariant-based MTS according to the traces.

图 9:根据轨迹细化基于不变量的中期战略。

tions tend to be more complex than the manually written ones [46].For example, consider a method postcondition that consists of a set of implication clauses that relate the program state before and after a method invocation: [(PreState1) ⇒ (PostState1)]∧...∧

操作往往比手工编写的复杂，[46]。例如，考虑一个方法后置条件，它由一组隐含子句组成，这些子句在方法调用前后关联程序状态:[(前置任务 1)(后置任务 1)]...∪

[(PreStaten) ⇒ (PostStaten)].The states in the TEMI's invariant-based MTS would correspond to the different states PreState1,..., PreStaten, and each state would have a transition to its appropri-ate next state PostState1,..., PostStaten.On the other hand, Con-tractor's model would have a single state corresponding to all the pre-states with nondeterministic transitions to the post-states, thus resulting in a less precise model.

[。基于 TEMI 不变量的中期战略中的状态将对应于不同的状态 1，...，前置状态，并且每个状态都将转换到其适当的下一状态后置状态 1，...，PostStaten。另一方面，Con-tractor 的模型会有一个对应于所有前状态的单一状态，并具有到后状态的不确定转换，从而导致模型不太精确。

Phase II: Refining the Invariant-based MTS

第二阶段:完善基于不变量的多边贸易体系

TEMI uses observed trace information to refine the invariant-based MTS by promoting to required those maybe transitions that corre-spond to observed invocations.REFINEINVARIANTMTS in Figure 9 describes this refinement algorithm.In [30], we summarize addi-tional optimizations introduced to improve the scalability of TEMI as well as extensions that add further information to the inferred models.We omit their discussion for brevity as they are not part of the core algorithm.

TEMI 使用观察到的跟踪信息来细化基于不变量的 MTS，方法是将那些可能对应于观察到的调用的转换提升到必需的状态。图 9 中的 REFINEINVARIANTMTS 描述了这种细化算法。在[30]中，我们总结了为提高 TEMI 的可伸缩性而引入的附加优化，以及为推断模型添加更多信息的扩展。为了简洁起见，我们省略了它们的讨论，因为它们不是核心算法的一部分。

182

182

S0

S0

S4' S3'

S4' S3 '

S1' push

S1 的推动

ET, FT, TN, TPN, Emp

ET、FT、TN、TPN、Emp

ET, FF, Emp, TN, TPN

ET、FF、Emp、TN、TPN

EF, FT, TV

英孚、英国《金融时报》、电视

StackAr

StackAr

StackAr

StackAr

S1"

S1”

ET, FF, Emp?, TN, TPN

外星人，FF，Emp？、总氮、主题方案网络

TPV, Emp

TPV，电磁脉冲

push

推

S1'"

“S1”

S2

S2

push, EF?, FF?, TV?, TPV

用力，英孚？，FF？电视？，TPV

push

推

ET?, FF?, Emp?, TN?, TPN?

外星人？，FF？，Emp？田纳西州？，主题方案网络?

S3" TPV

S3" TPV

push

推

EF?, FT?, TV?

英孚？英国《金融时报》？电视？

TPVEmp push Emp

TPVEmp 推送 Emp

Figure 10: The StackAr MTS refined with invocation traces.

图 10:用调用跟踪细化的堆栈结构。

A direct approach to incorporating trace information into the invariant-based MTS is to simulate the traces on the MTS and promote each traversed maybe transition (denoted with '?'on the label) to a required transition (without '?').However, this approach can result in imprecisions because states may be visited multiple times, and the produced model would not distinguish between the different visits.This can "stitch" together a required transition from one trace to a required transition from another trace, resulting in an invocation ordering that never occurred.For example, consider the direct refinement of the invariant-based MTS from Figure 8 based on the StackAr traces from Figure 3.The resulting MTS allows a spurious sequence in which — based on Trace 2 — push(x) from an empty stack (state S1) to a full stack (S3) is followed — based on Trace 3 — by topAndPop() to a partially full stack (S2).

将跟踪信息合并到基于不变量的中期战略中的一个直接方法是模拟中期战略上的跟踪，并促进每个遍历的可能过渡(用“？”表示)标签上)转换到所需的转换(不带“？”)。然而，这种方法会导致不精确，因为国家可能被多次访问，并且所产生的模型不会区分不同的访问。这可以将从一个跟踪到另一个跟踪的所需转换“缝合”在一起，导致从未发生的调用顺序。例如，考虑图 8 中基于不变量的 MTS 基于图 3 中的堆栈跟踪的直接细化。由此产生的 MTS 允许一个伪序列，在该序列中，基于轨迹 2，将(x)从空堆栈(状态 S1)推(x)到满堆栈(S3)，然后基于轨迹 3，将 topAndPop()推(x)到部分满堆栈(S2)。

To avoid such issues, REFINEINVARIANTMTS enhances the direct refinement strategy by also refining the visited states (the refinement process is captured in lines 2-6).When REFINEINVARI-ANTMTS visits a state in the invariant-based MTS, it first splits it into two states (line 5, where REFINESTATE, shown in the bottom portion of Figure 9, is called).The first refined state (nextStin RE-FINESTATE) has only one incoming transition, which corresponds to the currently processed trace invocation (currentEv).The second refined state (nextSt) keeps the remaining incoming transitions of the original state.Each state keeps all of the original outgoing tran-sitions and self-transitions (lines 5-7 in REFINESTATE).The state splitting enables each state to express different behavior according to the incoming transitions.For example, StackAr's MTS depicted in Figure 10 is obtained by refining the invariant-based MTS from Figure 8 with the traces from Figure 3.The push(x) invocation from S5 to S6 in Trace 2 (Figure 3) splits StackAr's invariant-based state S3 (Figure 8) into two new states S 3 and S 3 (Figure 10).S 3 is reachable from empty-stack states, while S 3 is reachable from a par-tially full stack.Furthermore, the transition S 1 push $-\rightarrow_r$ S 3 is promoted to required since it has been observed in the traces.

为了避免这样的问题，REFINEINVARIANTMTS 还通过细化访问状态来增强直接细化策略(细化过程在第 2-6 行中捕获)。当 REFINEVARI-ANTMS 访问基于不变量的 MTS 中的一个状态时，它首先将它分成两个状态(第 5 行，这里调用 REFINEINVARI，如图 9 底部所示)。第一个细化状态(nextStin RE-

FINESTATE)只有一个传入转换，它对应于当前处理的跟踪调用(currentEv)。第二个细化状态(nextSt)保持原始状态的剩余输入转换。每个状态保留所有原始输出转换和自转换(REFINESTATE 中的第 5-7 行)。状态分割使每个状态能够根据输入的转换来表达不同的行为。例如，图 10 中描述的 StackAr 的 MTS 是通过用图 3 中的轨迹细化图 8 中基于不变量的 MTS 而获得的。跟踪 2(图 3)中从 S5 到 S6 的推(x)调用将 StackAr 的基于不变的状态 S 3(图 8)分成两个新的状态 S 3 和 S3(图 10)。S 3 可以从空栈状态到达，而 S 3 可以从部分满栈到达。此外，由于在走线中观察到转变 S 1 推动→R3，因此转变 S 3 被提升为必需。

Once the MTS is refined according to the traces, REFINEINVARI-ANTMTS uses the newly incorporated required transitions to further improve the model.First, REFINEINVARIANTMTS removes spuri-ous nondeterministic transitions for each method, using a heuristic that if a state has non-deterministic transitions on a method, and some of those transitions are observed (required) while others are un-observed (maybe), then the unobserved transitions can be removed (lines 7-9) because they are likely a product of overly general invari-ants.For example, StackAr's refined MTS in Figure 10 does not have a maybe transition from S 3 on topAndPop() to a partially full stack state S2.This is because such behavior was never observed.By contrast, a direct transition to an empty stack (S 3 topAndPop −→r S 1 ) was observed in Trace 2.This distinction between the refined states

一旦根据轨迹对 MTS 进行了细化，REFINEVARI-ANTMS 就使用新合并的所需转换来进一步改进模型。首先，REFINEINVARIANTMTS 为每种方法删除不确定的转换，使用一种启发式方法，如果一个状态在一种方法上有不确定的转换，并且其中一些转换被观察到(必需)，而另一些转换未被观察到(可能)，那么未观察到的转换可以被删除(第 7-9 行)，因为它们可能是过于一般的不变量的产物。例如，图 10 中 StackAr 的改进的 MTS 没有从 So3 到 Pop()到部分满栈状态 S2 的可能过渡。这是因为这种行为从未被观察到。相比之下，在跟踪 2 中观察到了向空堆栈的直接转换(S 3 到 AndPop→R1)。这种精炼状态之间的区别

Application Type Exec.Libraries

应用程序类型 Exec。图书馆

StackArTester [15] unit test unit test StackAr jEdit [26] text editor end-user StringTokenizer jlGUI [28] media player end-user StringTokenizer Columba [] e-mail client end-user SignatureSocketSMTPProtocol

StackArTester [15]单元测试单元测试 StackAr jEdit [26]文本编辑器最终用户字符串化器 jlGUI [28]媒体播放器最终用户字符串化器 Columba []电子邮件客户端最终用户签名回复邮件协议

jFTP [] file transfer end-user SignatureSftpConnection JarInstaller [25] packaging end-user ZipOutputStream

jFTP []文件传输终端用户签名连接 JarInstaller [25]打包终端用户压缩输出流

DaCapo Xalan [12] benchmark

达卡波·萨兰·[12]基准

perfor- NumberFormatStr-mance ingTokenizer, test ToHTMLStream

performat-Number format 字符串-启用生成器，测试到 TMLStream

Voldemort [] distributed unit tests Socket database

分布式单元测试套接字数据库

Figure 11: Eight applications that exercise the evaluated li-braries.

图 11:运行评估库的八个应用程序。

S 3 and S 3 was not present in the original invariant-based MTS due to topAndPop()'s incomplete postcondition.

由于 topAndPop()的不完全后置条件，S 3 和 S 3 不存在于原始的基于不变量的 MTS 中。

4.EVALUATION

4.估价

To evaluate the four model inference strategies, we used the four respective algorithms to infer models of nine open-source libraries.To generate the execution information necessary for our analysis, we exercised the libraries using other readily-available, open-source applications we found on the web.We then compared the quality of the models inferred by each of the four algorithms.Section 4.1 describes the applications and the libraries our evaluation uses.Sec-tion 4.2 explains the setup and goals of our evaluation.Sections 4.3, 4.4, and 4.5 present the evaluation results.Finally, Section 4.6 addresses the threats to the validity of our findings.

为了评估这四种模型推断策略，我们使用了四种相应的算法来推断九个开源库的模型。为了生成我们分析所需的执行信息，我们使用了我们在网上找到的其他容易获得的开源应用程序来运行库。然后，我们比较了四种算法中每一种所推断的模型的质量。第 4.1 节描述了我们评估使用的应用程序和库。第 4.2 节解释了我们评估的设置和目标。第 4.3、4.4 和 4.5 节介绍了评估结果。最后，第 4.6 节讨论了对我们调查结果有效性的威胁。

4.1 Subject Libraries and Applications

4.1 主题库和应用

Our nine evaluation libraries span five categories: 1.Data structures (DataStructures.StackAr) 2-3.Data processing (org.apache.xalan.templates.Elem-

我们的九个评估库跨越五个类别:1。数据结构(数据结构。StackAr) 2-3。数据处理(org . Apache . Xalan . templates . Elem-

Number.NumberFormatStringTokenizer, to which we will refer as NFST, and java.util.StringTokenizer) 4.Authentication and data-integrity verification (java.security.Signature)

号码。数字格式字符串化器，我们称之为 NFST，和 java.util.StringTokenizer) 4。认证和数据完整性验证(java.security .签名)

5-6.Data streaming (java.util.zip.ZipOutputStream and org.apache.xml.serializer.ToHTMLStream) 7-9.Distributed communication and message exchange

5-6。数据流(Java . util . zip . ZipOutStream 和 org . Apache . XML . serializer . ToTMlsStream)7-9。分布式通信和消息交换

(org.columba.ristretto.smtp.SMTPProtocol, net.sf.jftp.net.wrappers.SftpConnection, and java.net.Socket)

(org . columba . ristretto . SMTP . SMtpProtocol、net . SF . jftp . net . wrappers . SFtpconnection 和 java.net.Socket)

To collect invocation traces for these libraries, we used eight open-source applications.Figure 11 describes the applications and the way we ran them to exercise the libraries.For example, we exercised StringTokenizer's functionality by running jEdit as an end-user who edits and saves text, and Socket's functionality by running Voldemort's unit tests that involve socket-based communication.We provided all techniques with the same set of traces and inferred invariants as inputs.The collected traces contained all invocations of the selected classes.

为了收集这些库的调用跟踪，我们使用了八个开源应用程序。图 11 描述了应用程序以及我们运行它们来运行库的方式。例如，我们通过作为编辑和保存文本的最终用户运行 jEdit 来实现 StringTokenizer 的功能，通过运行涉及基于套接字通信的伏地魔单元测试来实现套接字的功能。我们为所有技术提供了相同的轨迹集和推断不变量作为输入。收集的跟踪包含所选类的所有调用。

4.2 Evaluation Setup

4.2 评估设置

This section describes our quality metrics, the process for assess-ing a model's quality based on ground-truth models, our hypotheses,

本节描述了我们的质量度量标准，基于基本事实模型评估模型质量的过程，我们的假设，



183

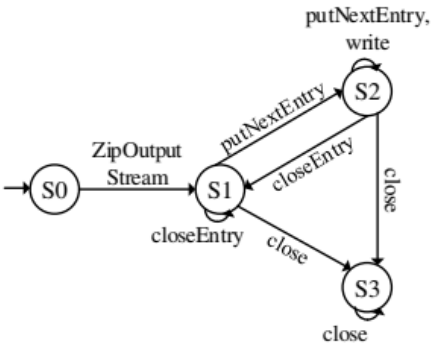


Figure 12: ZipOutputStream's ground-truth model.

图 12:ZipOutStream 的基本事实模型。

and our experiments.

我们的实验。

4.2.1 Metrics

4.2.1 指标

To measure the quality of a model, we compare it to a ground-truth model (see Section 4.2.2).We perform this comparison in terms of precision and recall [36].Precision measures the fraction of one thousand traces generated by the inferred model that are allowed by the ground-truth.Recall measures the fraction of one thousand ground-truth traces that are allowed by the inferred model, suggesting how complete the model is.Since the evaluated models can have infinite traces, we restricted the length of the traces to twice the number of transitions in the ground-truth model.

为了测量模型的质量，我们将其与基本事实模型进行比较(见第 4.2.2 节)。我们在精确度方面进行比较，并回忆[36]。精度测量由推理模型产生的一千条轨迹中地面真实所允许的部分。回忆测量推断模型允许的一千条地面真实轨迹的分数，表明模型有多完整。因为被评估的模型可以有无限的迹线，所以我们将迹线的长度限制为地面真实模型中跃迁次数的两倍。

4.2.2 Ground-Truth Models

4.2.2 基本事实模型

Evaluating the quality of the inferred models requires a set of ground-truth models that represent the libraries' legal behavior.To this end, we used the ground-truth models that were manually ex-tracted as a part of related work [16, 47].We modified those models when we discovered imprecisions that we were able to validate by in-specting the source code, and when the models included non-public methods.We removed from the ground-truth models all transi-tions on methods that were never invoked in the collected traces.(This simplification equally impacts the recall of our techniques and of the existing techniques, and thus does not affect our conclu-sions.Meanwhile, the precision of the techniques is unaffected as the removed methods are not present in the inferred models.) For libraries without an existing ground-truth (NFST, ToHTMLStream, and SftpConnection), two doctoral students inspected the source code and API documentation, and manually constructed the mod-els.As an example, Figure 12 depicts the ground-truth models for ZipOutputStream.

评估推断模型的质量需要一套代表图书馆法律行为的基本事实模型。为此，我们使用了作为相关工作的一部分手动提取的基本真理模型[16，47]。当我们发现不精确时，我们修改了这些模型，我们可以通过查看源代码来验证这些不精确，当这些模型包括非公共方法时。我们从基本事实模型中删除了在收集的踪迹中从未调用过的方法的所有转换。(这种简化同样会影响我们的技术和现有技术的回忆，因此不会影响我们的结论。同时，技术的精度不受影响，因为在推断的模型中不存在移除的方法。)对于没有现有基本事实的图书馆(NFST、东京都和旧金山图书馆联盟)，两名博士生检查了源代码和应用编程接口文档，并手动构建了模型。作为一个例子，图 12 描述了 ZipOutputStream 的基本事实模型。

4.2.3 Hypotheses

4.2.3 假设

Our evaluation tests the following hypotheses:

我们的评估测试了以下假设:

Hypothesis 1: Models inferred using invariants-only and trace-en-

假设 1:模型推断使用不变量-仅和跟踪-en-

hanced-invariants strategies are of significantly higher qual-ity as compared to models inferred using traces-only and invariant-enhanced-traces strategies.

与使用仅迹线和不变增强迹线策略推断的模型相比，增强不变量策略具有显著更高的质量。

Hypothesis 2: Execution traces can circumvent imprecisions stem-ming from invariant inference that may be incomplete.Hypothesis 3: Invariant filtering is a necessary aspect of invariants-based dynamic model inference techniques.

假设 2:执行跟踪可以绕过源于不变推理的不精确性，不变推理可能是不完整的。假设 3:不变量过滤是基于不变量的动态模型推理技术的一个必要方面。

4.2.4 Experimental Design

4.2.4 实验设计

For each library, we inferred seven models: traditional k-tail and SEKT with k ∈ {1,2}, optimistic and pessimistic TEMI, and CONTRACTOR++.The pessimistic TEMI model removes maybe transitions (thus treating unobserved invocations as illegal), while the optimistic model retains all transitions.We do not report results of k-tail for k > 2 because those k values led to fewer merges, which lowered the recall without notable precision improvement.

对于每个库，我们推断出七个模型:传统的 k-tail 和带有 k∑{ 1，2}的 SEKT，乐观和悲观的 TEMI，以及 CONTRACTOR++。悲观的 TEMI 模型删除了可能的转换(因此将未观察到的调用视为非法)，而乐观的模型保留了所有的转换。我们不报告 k > 2 的 k-tail 结果，因为这些 k 值导致更少的合并，这降低了召回率，但没有显著的精度提高。

In the experiments, for each library, we performed three steps: (1) execute applications that use the library to generate traces, (2) run the seven inference algorithms on those traces, and (3) measure the precision and recall of the inferred models.The implementa-tion, observed traces, inferred models, and ground-truth models are publicly available: http://softarch.usc.edu/wiki/doku. php?id=inference:start.

在实验中，对于每个库，我们执行三个步骤:(1)执行使用库生成轨迹的应用程序，(2)在这些轨迹上运行七个推理算法，以及(3)测量推理模型的精度和召回率。实现、观察到的轨迹、推断模型和基本事实模型都是公开的:http://softarch.usc.edu/wiki/doku. PHP？id =推论:开始。

4.3 Inferred Model Precision and Recall

4.3 推断模型精度和召回率

This section assesses the quality of the inferred models, as com-pared to the ground truth.Figure 13 shows the precision (P) and recall (R) of each algorithm's models.Figure 13 distinguishes the algorithms along two dimensions: The algorithms developed fully as part of our research, placed in the central portion of the table, are separated from the existing algorithms by the two vertical lines;the different shadings distinguish implementations of traces-only and invariant-enhanced-traces strategies from the algorithms that implement invariants-only and trace-enhanced-invariants strategies.Next, we (1) compare the models' precision and recall;(2) analyze the reasons for better recall of models inferred via the invariants-only and trace-enhanced-invariants strategies;and (3) explain the differences between models inferred using those two strategies.

本节评估了推断模型的质量，与基本事实进行了比较。图 13 显示了每个算法模型的精确度和召回率。图 13 沿着两个维度区分算法:作为我们研究的一部分而完全开发的算法，放在表格的中心部分，通过两条垂直线与现有算法分开；不同的阴影将仅跟踪和不变增强跟踪策略的实现与实现仅不变量和跟踪增强不变

量策略的算法区分开来。接下来，我们(1)比较模型的精度和召回率；(2)分析通过仅不变量和跟踪增强不变量策略推断的模型的更好召回的原因；和(3)解释使用这两种策略推断的模型之间的差异。

For all libraries, except StringTokenizer, CONTRACTOR++ and TEMI produce models of superior recall and comparable or better precision than the traditional k-tail and SEKT algorithms.For example, the k-tail models of SMTPProtocol allow only a single message to be sent before terminating the connection.The TEMI model correctly generalizes the observed behavior and allows multi-ple messages, which improves the recall.Compared with SEKT, for example, the precision of the optimistic TEMI is lower in the cases where TEMI either included an erroneous transition or inferred erro-neous states due to incomplete invariants.Pessimistic TEMI does resolve the imprecisions due to erroneous transitions by considering only required transitions.

对于所有库，除了 StringTokenizer、CONTRACTOR++和 TEMI，生成的模型比传统的 k-tail 和 SEKT 算法具有更高的召回率和可比较或更高的精度。例如，SMTPProtocol 的 k 尾模型只允许在终止连接之前发送一条消息。TEMI 模型正确地概括了观察到的行为，允许多条消息，提高了召回率。例如，与 SEKT 相比，在 TEMI 由于不完全不变量而包含错误跃迁或推断的错误状态的情况下，乐观 TEMI 的精度较低。悲观的 TEMI 通过只考虑所需的转换来解决由于错误转换而导致的不精确性。

Figure 13 indicates that TEMI and CONTRACTOR++, have sig-nificantly higher recall than the k-tail-based algorithms.This is because the invariants help to distinguish between those invocations that change program state, in turn restricting future invocations, and those that do not.For example, ToHTMLStream's invariants indicate no restrictions on its method invocations and the CONTRACTOR++ model has a single state with a self-transition for every method.In contrast, the k-tail models capture irrelevant invocation restrictions inferred from the traces.Furthermore, the results confirm that, while the traces-only strategy has high precision on average, its complete dependence on execution traces can result in precision-lowering spurious merges (e.g., this happened for models of StackAr and SftpConnection).

图 13 表明，TEMI 和 CONTRACTOR++的召回率明显高于基于 k-tail 的算法。这是因为不变量有助于区分改变程序状态的调用和不改变程序状态的调用。例如，ToHTMLStream 的不变量表示对其方法调用没有限制，CONTRACTOR++模型有一个单独的状态，每个方法都有一个自转换。相比之下，k-tail 模型捕获从跟踪中推断出的无关调用限制。此外，结果证实，尽管仅跟踪策略平均具有高精度，但它对执行跟踪的完全依赖会导致精度降低的虚假合并(例如，StackAr 和 SftpConnection 模型就发生了这种情况)。

Our evaluation results also suggest that TEMI is slightly-to-mod-erately more precise than CONTRACTOR++, while having nearly-identical recall.The differences in precision stem from the ways these two approaches construct model states and the way TEMI incorporates the observed invocations.As discussed in Section 3.4, multiple TEMI states may be logically mapped to a single CON-TRACTOR++ state;the CONTRACTOR++ state may have additional, precision-lowering, transitions that do not exist in the TEMI states.

我们的评估结果还表明，TEMI 比 CONTRACTOR++略精确到适度，但召回率几乎相同。精度上的差异源于这两种方法构建模型状态的方式和 TEMI 合并观察到的调用的方式。如第 3.4 节所讨论的，多个 TEMI 状态可以逻辑映射到一个 CON-TRACTOR++状态；承包商++状态可能有 TEMI 状态中不存在的额外的、精度降低的转换。

The difference in precision is especially pronounced for String-Tokenizer and SMTPProtocol, whose recall CONTRACTOR++ im-proved by up to 1%, but at a precision cost of 7-8%, as compared to optimistic TEMI.CONTRACTOR++'s model of StringTokenizer allowed illegal invocation sequences in which a first invocation of hasMoreTokens() returned false but a

subsequent invocation returned true.This occurred because CONTRACTOR++ is not al-ways able to precisely capture postconditions that relate post-state

对于字符串令牌化器和 SMTPProtocol 来说，精度上的差异尤其明显，它们的召回率 CONTRACTOR++提高了 1%，但与乐观的 TEMI 相比，精度成本提高了 7-8%。CONTRACTOR++的 StringTokenizer 模型允许非法调用序列，其中第一次调用 hasMoreTokens()返回 false，但后续调用返回 true。出现这种情况是因为 CONTRACTOR++无法精确捕获与后置状态相关的后置条件



184

| Library | Traditional 1-tail | | Traditional 2-tail | | SEKT 1-tail | | SEKT 2-tail | | Optimistic TEMI | | Pessimistic TEMI | | CONTRACTOR++ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | P | R | P | R | P | R | P | R | P | R | P | R | P | R |
| StackAr | 64% | 30% | 83% | 30% | 97% | 30% | 97% | 30% | 99% | 94% | 100% | 71% | 99% | 94% |
| NFST | 100% | 21% | 100% | 21% | 100% | 21% | 100% | 21% | 96% | 57% | 96% | 44% | 96% | 57% |
| StringTokenizer | 100% | 52% | 100% | 51% | 100% | 51% | 100% | 50% | 100% | 51% | 100% | 50% | 93% | 52% |
| Signature | 100% | 61% | 100% | 61% | 100% | 61% | 100% | 61% | 100% | 88% | 100% | 88% | 100% | 88% |
| ToHTMLStream | 100% | 26% | 100% | 26% | 100% | 26% | 100% | 26% | 100% | 100% | 100% | 100% | 100% | 100% |
| ZipOutputStream | 100% | 37% | 100% | 36% | 100% | 37% | 100% | 35% | 100% | 63% | 100% | 47% | 100% | 63% |
| SMTPProtocol | 100% | 20% | 100% | 20% | 100% | 20% | 100% | 20% | 96% | 77% | 100% | 66% | 88% | 78% |
| Socket | 94% | 24% | 97% | 21% | 100% | 23% | 100% | 21% | 100% | 67% | 100% | 51% | 100% | 67% |
| SftpConnection | 61% | 31% | 96% | 29% | 100% | 30% | 100% | 29% | 97% | 48% | 100% | 33% | NA | NA |
| **Average** | 95% | 34% | 98% | 33% | 100% | 34% | 100% | 33% | 99% | 75% | 99% | 65% | 97% | 75% |

Figure 13: Precision (P) and recall (R) comparison of k-tail, SEKT, TEMI, and the CONTRACTOR++ (enhanced Contractor) al-gorithms.The Average row excludes results for SftpConnection, because CONTRACTOR++ ran out of memory during model generation.

图 13:k-tail、SEKT、TEMI 和承包商++(增强型承包商)算法的精度和召回率比较。“平均值”行排除了 SftpConnection 的结果，因为 CONTRACTOR++在模型生成期间内存不足。

values to pre-state values.For SMTPProtocol, TEMI removed un-observed transitions on getState(), which remained in the CON-TRACTOR++ model due to incomplete invariants.

值转换为预状态值。对于 SMTPProtocol，TEMI 删除了 getState()上未观察到的转换，由于不变量不完整，这些转换仍然保留在 CON-TRACTOR++模型中。

In general, the causes behind CONTRACTOR++'s imprecision were more varied than those that impacted the recall of TEMI: While TEMI's recall may deteriorate because of overly restrictive invari-ants abstracted by CONTRACTOR++, CONTRACTOR++'s precision may be hampered by incomplete invariants, intricate relationships between invariants, and invocation dependencies that invariants can-not capture.We note that, while issues such as overly restrictive or incomplete invariants can be mitigated by collecting additional exe-cutions, invariant complexity and implied invocation dependencies cannot be mitigated in such a way.

总的来说，承包商++不精确背后的原因比那些影响 TEMI 召回事件的原因更为多样:尽管 TEMI 召回事件可能会因为承包商++提取的过度限制性不变量而恶化，但承包商++的精度可能会受到不完全不变量、不变量之间复杂关系以及不变量无法捕获的调用依赖关系的影响。我们注意到，虽然收集额外的执行可以缓解过度限制或不完整的不变量等问题，但不变的复杂性和隐含的调用依赖不能以这种方式缓解。

Our results strongly support Hypothesis 1 from Section 4.2: Uti-lizing program state information from invariants to create an initial model, before potentially augmenting it with information about invocation sequences from the execution traces, signifi-cantly improves the quality of dynamically inferred models.Our results also suggest that combining program state information with information about invocation sequences from execution traces results in near perfectly-precise models, as was the case for our invariant-enhanced-traces (SEKT) and trace-enhanced-invariants (TEMI) algorithms.

我们的结果强烈支持第 4.2 节中的假设 1:利用不变量中的程序状态信息创建一个初始模型，然后用来自执行轨迹的调用序列信息对其进行潜在的扩充，显著提高了动态推断模型的质量。我们的结果还表明，将程序状态信息与来自执行跟踪的调用序列信息相结合，可以得到近乎完美精确的模型，就像我们的不变增强跟踪(SEKT)和跟踪增强不变(TEMI)算法一样。

4.4 Sensitivity to Invariant Quality

4.4 对不变质量的敏感性

In the real world, the collected execution data may be partial, and the results of invariant inference noisy.Model inference's aim is to maintain high precision under noise: Even a moderate drop in precision could render the produced model useless [48].

在现实世界中，收集的执行数据可能是部分的，并且不变推理的结果有噪声。模型推断的目的是在噪音下保持高精度:即使精度的适度下降也会使产生的模型变得无用[48]。

To evaluate the impact of invariant noise, we removed random subsets of invariant clauses and measured the resulting precision and recall of the TEMI and CONTRACTOR++ models.For each library, we generated 20 models for each of the cases when 10% and 20% of the invariant clauses are removed.The results in Fig-ure 14 suggest that (1) the precision of pessimistic TEMI is robust to variations in invariant quality, (2) optimistic TEMI models out-perform CONTRACTOR++ by yielding higher precision and recall, (3) noisy environments require enhancing invariants with trace in-formation, and (4) decreased invariant quality negatively affects CONTRACTOR++'s performance.We elaborate on each point next.

为了评估不变噪声的影响，我们移除了不变子句的随机子集，并测量了 TEMI 和承包商++模型的结果精度和召回率。对于每个库，当 10%和 20%的不变子句被删除时，我们为每个案例生成了 20 个模型。图 14 中的结果表明:(1)悲观 TEMI 的精度对不变质量的变化是稳健的，(2)乐观 TEMI 模型通过产生更高的精度和召回率而超过承包商++，(3)噪声环境需要用跟踪信息增强不变量，以及(4)不变质量的降低对承包商++的性能产生负面影响。接下来我们详细阐述每一点。

1.Invariant incompleteness did not affect the near-perfectly pre-cise (99%) pessimistic TEMI models.This confirms that the MTS refinement procedure (recall Section 3.4) appropriately introduces the trace information even when the initial model is imperfect.In particular, pessimistic TEMI models have 8-13% higher precision

1.不变的不完全性并不影响近乎完美的预测(99%)悲观的 TEMI 模型。这证实了即使初始模型不完善，MTS 细化过程(回忆第 3.4 节)也适当地引入了跟踪信息。尤其是悲观的 TEMI 模型精度高 8-13%

than optimistic TEMI and CONTRACTOR++ models, while also having 33% higher recall than comparably precise k-tail models, which are not affected by noisy invariants (see Figure 13).

比乐观的 TEMI 模型和承包商++模型，同时也比没有噪声不变量影响的相对精确的 k 尾模型有 33%的高召回率(见图 13)。

2.When faced with noisy invariants, the optimistic TEMI models outperform CONTRACTOR++ in the average case by 1-4% in both precision and recall.The reasons were twofold: (1) Incomplete invariants add erroneous nondeterministic transitions;the degree of nondeterminism is higher in CONTRACTOR++ models, leading to lower precision.(2) Incomplete invariants cause TEMI's refinement procedure to correctly split states and subsequently remove the undesired maybe transitions only from the appropriate state.

2.当面对有噪声的不变量时，乐观的 TEMI 模型在精确度和召回率方面比承包商++平均高出 1-4%。原因有两个:(1)不完全不变量增加了错误的不确定转换；承包商++模型中的不确定性程度较高，导致精度较低。(2)不完全不变量导致 TEMI 的精化过程正确地分裂状态，并随后仅从适当的状态中移除不期望的可能转变。

3.In the average case, noisy invariants caused the TEMI models' precision to drop from 99% (Figure 13) to 89-91% (Figure 14);the precision dropped by 42% and 33% in the two extreme cases — StackAr and Socket.A reason for the extremes is that the states in TEMI's Socket model can have a number of incorrect nondeter-ministic transitions that confirm the connection (isConnected()= true) before it has been established.This highlights the crucial role of augmenting invariants-only models with trace information: isConnected() never returns this result in the actual traces and the erroneous transitions do not exist in the pessimistic TEMI models.

3.在一般情况下，噪声不变量导致 TEMI 模型的精度从 99%(图 13)下降到 89-91%(图 14)；在 StackAr 和 Socket 这两种极端情况下，精度分别下降了 42%和 33%。出现这种极端情况的一个原因是，TEMI 套接字模型中的状态可能有许多不正确的非管理转换，这些转换在连接建立之前会确认连接(isConnected()= true)。这突出了用跟踪信息增强仅不变量模型的关键作用:isConnected()在实际跟踪中从不返回这个结果，并且在悲观的 TEMI 模型中不存在错误的转换。

4.Incomplete invariants affect performance, making CONTRAC-TOR++ less efficient due to a higher number of allowed program states.This, in turn, results in a higher number of SMT queries that cause the scalability problems further discussed in Section 4.5.

4.不完全不变量影响性能，由于允许的程序状态数量较多，使得 CONTRAC-TOR++效率较低。这反过来又会导致更多的 SMT 查询，从而导致第 4.5 节中进一步讨论的可伸缩性问题。

We conclude that invocation trace information is necessary to circumvent potential imprecisions in the invariants-only models.Hypothesis 2 holds for trace-enhanced-invariants models, as less reliable inferred program invariant information is augmented with information about invocation sequences from the execution traces.While lower quality invariants reduce the precision of the maybe transitions, the pessimistic TEMI models remained almost perfectly precise with unchanged recall, which makes pessimistic TEMI the most appropriate choice when an engineer is running an unfamiliar system to infer a model of a poorly documented library.

我们得出结论，调用跟踪信息对于规避仅不变量模型中的潜在不精确性是必要的。假设 2 适用于跟踪增强不变量模型，因为不太可靠的推断程序不变量信息被来自执行跟踪的调用序列的信息所增强。虽然较低质量的不变量降低了可能转换的精度，但悲观的 TEMI 模型几乎保持了完美的精度，召回率没有变化，这使得当工程师运行一个不熟悉的系统来推断一个文档不足的库的模型时，悲观的 TEMI 是最合适的选择。

4.5 Impact of Invariant Filtering

4.5 不变滤波的影响

As noted earlier, the high quality of invariants-only models is critically dependent on having meaningful and manageable invari-ants as inputs.This is best described using the differences between the original Contractor algorithm [18] and CONTRACTOR++, our enhancement of Contractor with invariant filters (recall Section 3.2).

如前所述，高质量的只含不变量的模型非常依赖于有意义和可管理的不变量作为输入。最好使用原始承包商算法[18]和承包商++(我们用不变过滤器增强承包商)之间的差异来描述这一点(回忆第 3.2 节)。

Figure 15 outlines the model quality when the enhancements are not applied.Compared to CONTRACTOR++, when the input invari-

图 15 概述了没有应用增强时的模型质量。与 CONTRACTOR++相比，当输入因瓦时-



185

| Library | Optimistic TEMI | | Optimistic TEMI | | Pessimistic TEMI | | CONTRACTOR++ | | Optimistic TEMI | | Pessimistic TEMI | | CONTRACTOR++ | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | full invariant set | | 10% invariants removed | | | | | | 20% invariants removed | | | | | |
| | P | R | P | R | P | R | P | R | P | R | P | R | P | R |
| StackAr | 99% | 94% | 67% | 96% | 100% | 71% | 70% | 95% | 66% | 96% | 100% | 80% | 61% | 97% |
| NFST | 96% | 57% | 90% | 59% | 96% | 44% | 86% | 60% | 86% | 60% | 96% | 44% | 82% | 69% |
| StringTokenizer | 100% | 51% | 90% | 77% | 100% | 50% | 92% | 69% | 82% | 79% | 100% | 50% | 91% | 72% |
| Signature | 100% | 88% | 96% | 90% | 100% | 88% | 96% | 88% | 92% | 91% | 100% | 88% | 88% | 77% |
| ToHTMLStream | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% | 100% |
| ZipOutputStream | 100% | 63% | 100% | 69% | 100% | 47% | 100% | 69% | 100% | 76% | 100% | 47% | 100% | 73% |
| SMTPProtocol | 96% | 77% | 96% | 77% | 100% | 66% | 86% | 74% | 94% | 78% | 100% | 66% | 80% | 62% |
| Socket | 100% | 67% | 61% | 69% | 100% | 51% | NA | NA | 58% | 71% | 100% | 51% | NA | NA |
| **Average** | 99% | 76% | 91% | 81% | 99% | 67% | 90% | 79% | 89% | 83% | 99% | 67% | 86% | 79% |

Figure 14: Precision (P) and recall (R) comparison of TEMI, and the CONTRACTOR++ algorithms in a noisy environment (when some of the invariants are removed).TEMI is more robust to the noise, with almost-perfectly precise pessimistic TEMI, while delivering slightly higher recall on average.The Average excludes results for Socket, because CONTRACTOR++ ran out of memory during model generation.

图 14:TEMI 和承包商++算法在嘈杂环境中(当一些不变量被去除时)的精度和召回率比较。TEMI 对噪音更加稳健，几乎完美精确的悲观 TEMI，同时平均召回率略高。平均值排除了套接字的结果，因为承包商++在模型生成期间内存不足。

ants are not filtered (No Invariant Filtering in Figure 15), a modest 1% average increase in precision comes at the expense of a sizable drop in recall (up to 54%, in the case of SMTPProtocol).When the different method return points are not handled separately (No Method Distinction), the resulting models are 25% less precise on average, although the average recall increases by 10%.Omitting our enhancements also accentuated Contractor's scalability problems, denoted as NA values, discussed next.

蚂蚁没有被过滤(图 15 中没有不变过滤)，精度平均提高 1%，代价是召回率大幅下降(在 SMTPProtocol 的情况下，高达 54%)。当不同的方法返回点没有单独处理时(无方法区分)，结果模型平均精度降低 25%，尽管平均召回率增加了 10%。省略我们的增强还会加重承包商的可伸缩性问题，表示为 NA 值，下面讨论。

Each CONTRACTOR++ SMT query includes the invariants of all methods, and such a query is generated for every possible combina-tion of methods' invariant evaluations (recall Section 3.2).Hence, CONTRACTOR++ queries are longer and more resource consuming than TEMI's queries.In the case of SftpConnection, which has 22 methods with 684 invariant clauses, the SMT solver runs out of memory (Figure 13);this happens in five additional cases when our enhancements are not applied (Figure 15), as well as the one case with noisy invariants (Figure 14).Since CONTRACTOR++ is built using the combination of Python and C, the memory is allocated by the operating system, which kills the SMT solver's process once the memory consumption makes the system unstable.

每个承包商++ SMT 查询都包含所有方法的不变量，并且这种查询是针对方法的不变量评估的每种可能组合而生成的(回忆第 3.2 节)。因此，承包商++查询比 TEMI 的查询更长，消耗更多资源。在 SftpConnection 的情况下，它有 22 个方法和 684 个不变子句，SMT 求解器内存不足(图 13)；当我们的增强没有被应用时，这发生在另外五种情况下(图 15)，以及一种有噪声不变量的情况下(图 14)。由于 CONTRACTOR++是使用 Python 和 C 的组合构建的，内存由操作系统分配，一旦内存消耗使系统不稳定，操作系统就会终止 SMT 求解程序的进程。

Overall, our evaluation results confirm Hypothesis 3, as invariant filters that keep only a limited set of relevant invariant types have shown to be crucial to enhancing the scalability as well as the quality of techniques that implement invariants-only and trace-enhanced-invariants strategies.

总的来说，我们的评估结果证实了假设 3，因为只保留一组有限的相关不变量类型的不变量过滤器对于提高实现仅不变量和跟踪增强不变量策略的技术的可伸缩性和质量至关重要。

4.6 Threats to Validity

4.6 有效性威胁

We now outline the threats to our evaluation's validity and discuss our mitigation strategies.

我们现在概述评估有效性面临的威胁，并讨论我们的缓解策略。

Library CONTRACTOR++ No Invariant No Method Filtering Distinction

库承包商++无不变量无方法过滤区分

P R P R P R

公关公关公关

StackAr 99% 94% 99% 94% 77% 100%

StackAr 99% 94% 99% 94% 77% 100%

NFST 96% 57% 98% 40% 78% 66%

NFST 96% 57% 98% 40% 78% 66%

StringTokenizer 93% 52% 91% 77% 47% 77%

字符串化器 93% 52% 91% 77% 47% 77%

Signature 100% 88% 100% 78% NA NA

签名 100% 88% 100% 78% NA

ToHTMLStream 100% 100% 100% 100% NA NA

至 TMLStream 100% 100% 100% 100% NA

ZipOutputStream 100% 63% NA NA NA NA

zip outputstream 100% 63% NA NA NA NA NA NA

SMTPProtocol 88% 78% 100% 24% NA NA

SMTPProtocol 88% 78% 100% 24% NA

Socket 100% 67% 96% 60% 86% 68%

插座 100% 67% 96% 60% 86% 68%

Figure 15: Precision (P) and recall (R) of Contractor models with and without SEKT- and TEMI-specific invariant filters.

图 15:带有和不带有特定于东南福溪和 TEMI 的不变滤波器的承包商模型的精度和召回率。

Ground-truth bias.The ground-truth models were, in part, man-ually constructed.This may make them biased to the specifier's expertise.To mitigate this threat, we used the publicly available ground-truths from other researchers [16, 47], and modified them only if we were able to validate the modifications by inspecting the source code.We also made multiple iterations over the resulting models with two specifiers.

基本事实偏见。基本真理模型部分是人为构建的。这可能会使他们偏向说明符的专业知识。为了减轻这种威胁，我们使用了其他研究人员[16，47]公开提供的基本事实，并且只有在我们能够通过检查源代码来验证修改的情况下才修改它们。我们还用两个说明符对结果模型进行了多次迭代。

Subject libraries and applications.The selection of libraries and applications that invoke them may bias the evaluation results.For instance, models of a well-known library may not be representa-tive of dynamically inferred models in general.Similarly, mature applications may be more careful in using a library's functionality.To this end, we selected libraries of different types and popularity (e.g., well documented Java libraries vs. less widely known NFST).We also used applications from several different domains and of different maturities, popularities, and sizes (e.g., widely used Volde-mort vs. less popular JarInstaller).Finally, we had to address the bias stemming from inferring models based on traces obtained from unit and integration tests.Hence, we collected some of the uti-lized traces by executing open-source applications in ways that an end-user would use them to explore the available features.

主题库和应用程序。选择调用它们的库和应用程序可能会影响评估结果。例如，众所周知的库的模型一般来说可能不是动态推断模型的代表。同样，成熟的应用程序在使用库的功能时可能会更加小心。为此，我们选择了不同类型和受欢迎程度的库(例如，记录良好的 Java 库与不太为人所知的 NFST 库)。我们还使用了来自多个不同领域、不同到期日、流行程度和规模的应用程序(例如，广泛使用的 Volde-mort 与

不太流行的 JarInstaller)。最后，我们必须解决基于从单元和集成测试中获得的轨迹推断模型所产生的偏差。因此，我们通过执行开源应用程序来收集一些实用化的跟踪，最终用户可以使用它们来探索可用的特性。

Metrics.There are multiple ways of comparing the generated models with the corresponding ground-truth models (e.g., recall and precision of the simulated traces vs. graph comparison), which could potentially yield different results.Our mitigation strategy used metrics that have been proposed and adopted by the research community [36], and are consistent with our goal of comparing model behavior, not structural similarity.

度量。有多种方法可以将生成的模型与相应的基本事实模型进行比较(例如，模拟轨迹的回忆和精度与图形比较)，这可能会产生不同的结果。我们的缓解策略使用了[36]研究团体提出并采用的指标，并且与我们比较模型行为而不是结构相似性的目标一致。

5.OUR FINDINGS' IMPACT

5.我们的发现的影响

While studying inferred model quality is an interesting exercise on its own, our findings, which confirm the high quality and robustness of trace-enhanced-invariants models, should motivate further appli-cation of model inference in practice.Below, we briefly elaborate on the utility of having the different types of generated models, and discuss the potential impact of our higher-quality trace-enhanced-invariants models on several development activities.

尽管研究推断模型质量本身是一项有趣的工作，但我们的发现证实了迹增强不变量模型的高质量和鲁棒性，这将推动模型推断在实践中的进一步应用。下面，我们简要阐述不同类型生成模型的效用，并讨论我们的高质量跟踪增强不变量模型对几个开发活动的潜在影响。

Model quality.The high overall quality of the inferred models can aid a variety of software development tasks including API under-standing [1, 18], debugging, test generation [41], and runtime fault detection [22, 48].For example, during debugging a programmer can analyze if a given library is invoked as expected.Similarly, an inferred trace-enhanced-invariants model that exhibits high recall (i.e., that is complete) can help to detect invocations that violate the library's protocol [48], while avoiding spurious warnings.

模型质量。推断模型的高总体质量可以帮助各种软件开发任务，包括应用编程接口下的[1，18]，调试，测试生成[41]，和运行时故障检测[22，48]。例如，在调试期间，程序员可以分析给定的库是否按预期被调用。类似地，显示高召回率(即完整)的推断的跟踪增强不变量模型可以帮助检测违反库的协议[48 的调用，同时避免虚假警告。



186

Required vs. maybe transitions.The TEMI models contain ob-served required transitions and unobserved maybe transitions.This dichotomy is useful for several reasons.For example, as discussed in Section 4.4, when the available executions are not comprehensive, the resulting invariants can be noisy.In such cases, a developer can rely on TEMI models due to almost perfect precision of the required transitions.In addition, the required MTS transitions can be augmented with frequencies of method invocations.This can benefit recommender systems [6, 7, 42, 50] in prioritizing examples by selecting those that commonly occur in actual executions.

必需的和可能的转换。TEMI 模型包含已观察到的所需过渡和未观察到的可能过渡。这种二分法之所以有用，有几个原因。例如，如第 4.4 节所讨论的，当可用的执行不全面时，产生的不变量可能会有噪声。在这种情况下，开发人员可以依赖 TEMI 模型，因为所需转换的精度几乎是完美的。此外，所需的 MTS 转换可以随着方法调用的频率而增加。这有利于推荐系统[6，7，42，50]通过选择那些在实际执行中经常出现的例子来对例子进行优先排序。

Program state information.While our evaluation focused on the invocation sequences, our models also relate the model states to the internal program states.Hence, instead of having to gain a comprehensive understanding of a library's source code or to strictly rely on the available API-level documentation, a programmer can use the state information to "peek inside" the library's implementa-tion.Similarly, tools that detect protocol violations (e.g., [48]) may provide more informative warnings by referring to program state.

程序状态信息。虽然我们的评估侧重于调用序列，但是我们的模型也将模型状态与内部程序状态联系起来。因此，程序员可以使用状态信息“窥视”库的实现，而不是必须全面理解库的源代码或严格依赖可用的 API 级文档。类似地，检测协议违反的工具(例如，[48])可以通过参考程序状态来提供更多信息警告。

Trace-oriented models.The invariant-enhanced-traces models can be more appropriate for development tasks that require trace-specific information, despite their significantly lower recall than that of the TEMI models.For example, detailed analysis of how an unfamiliar program communicates with a library requires compact, yet accurate representation of the traces themselves as opposed to a representation of the library's full API protocol.The SEKT algorithm produces such models, while the k-tail algorithm is less reliable due to its sole dependence on execution traces.

面向跟踪的模型。不变增强跟踪模型可能更适合于需要跟踪特定信息的开发任务，尽管它们的召回率比 TEMI 模型低得多。例如，对一个不熟悉的程序如何与一个库通信的详细分析需要对跟踪本身进行紧凑而精确的表示，而不是对库的完整的应用编程接口协议进行表示。SEKT 算法产生这样的模型，而 k-tail 算法不太可靠，因为它只依赖于执行轨迹。

## 6.RELATED WORK

## 6.相关著作

Program invariants have been used to directly synthesize FSM models [18], and to augment the k-tail algorithm with transition invariants [41].As noted in Sections 4.3 and 4.4, TEMI is more appropriate than CONTRACTOR++ for dynamic specification mining because (1) it has higher precision, (2) it is more resilient to invariant noise, and (3) unlike Contractor, it distinguishes between observed and unobserved invocations.By contrast, starting from the observed executions and using Daikon [20] to infer transition invariants [41] results in other imprecisions, as discussed in Section 3.3.

程序不变量已被用来直接合成有限状态机模型[18]，并用转换不变量[41]来增强 k 尾算法。如第 4.3 节和第 4.4 节所述，对于动态规范挖掘，TEMI 比承包商++更合适，因为(1)它具有更高的精度，(2)它对不变噪声更有弹性，(3)与承包商不同，它区分观察到的调用和未观察到的调用。相反，从观察到的执行开始，使用戴康[20]推断过渡不变量[41]会导致其他不精确，如第 3.3 节所讨论的。

The k-tail algorithm [5] serves as a basis for many FSM-inference techniques from invocation traces [10, 36, 37, 39, 41, 49, 56].These algorithms (1) extend k-tail to improve its precision or recall [10, 39, 49, 56], (2) build larger frameworks with k-tail as the inference algorithm [37, 49], and (3) enhance the models with information about invocation probabilities [37] and program state and method parameters [41].Additional merges can make the models more com-pact [10, 49], while

stricter merging conditions based on pairwise sequencing invariants can improve precision [39, 56].In general, these approaches' recall is only as good as the traditional 1-tail and, for our evaluation libraries, their precision cannot surpass the precision of SEKT.Synoptic [4], CSight [3], and Perfume [43, 44] use the CEGAR [8] approach to create a coarse initial model, and then refine it using counterexamples that falsify temporal invariants.InvariMint [2] presents a declarative specification language for ex-pressing model-inference algorithms, and improves the efficiency of algorithms, but neither their precision nor recall.

k-tail 算法[5]作为来自调用轨迹[10，36，37，39，41，49，56]的许多有限状态机推理技术的基础。这些算法(1)扩展 k-tail 以提高其精度或召回[10，39，49，56]，(2)用 k-tail 构建更大的框架作为推理算法[37，49]，和(3)用关于调用概率的信息增强模型[37]和程序状态和方法参数[41]。额外的合并可以使模型更加符合[10，49]，而基于成对排序不变量的更严格的合并条件可以提高精度[39，56]。总的来说，这些方法的召回率仅与传统的单尾方法一样好，并且对于我们的评估库来说，它们的精度不能超过 SEKT 的精度。天气[4]，[3]，香水[43，44]使用塞格[8]方法创建一个粗略的初始模型，然后使用伪造时间不变量的反例来改进它。因瓦明·[2]提出了一种声明性规范语言，用于扩展模型推理算法，提高了算法的效率，但既没有提高算法的精度，也没有提高算法的召回率。

There are other ways to aid development tasks than inferring models, such as detecting method invocation patterns [21, 60] and inter-object sequence charts [32, 38].These patterns can detect code anomalies [22, 48].Scenario-based views of a system's behav-ior, such as parametrized [38] and symbolic [32] sequence charts, facilitate understanding of a system's runtime interactions.

除了推断模型，还有其他方法来帮助开发任务，例如检测方法调用模式[21，60]和对象间序列图[32，38]。这些模式可以检测代码异常[22，48]。基于场景的系统行为视图，例如参数化的[序列图和符号化的[序列图，有助于理解系统的运行时交互。

While we used both CONTRACTOR++ and TEMI in combination with invariants inferred by Daikon [20], invariants-only and trace-

虽然我们同时使用了承包商++和 TEMI 以及由戴康·[20 推出的不变量，但不变量仅限于跟踪

enhanced-invariants techniques can be adapted to work with other invariant-inference techniques.Examples of such techniques include techniques that combine inference with symbolic program execu-tion [11] and enhance it by using simple, manually written initial invariants [57] or manually-written relations between methods [35].

增强不变量技术可以适用于其他不变量推理技术。这种技术的例子包括将推理与符号程序执行相结合的技术，并通过使用简单的、手动编写的初始不变量[57]或方法[35]之间的手动编写的关系来增强它。

Static [19, 53] and hybrid techniques [17, 58] provide two alter-natives to specification mining based strictly on invocation traces.Shoham et al. [53] infer, from client-side code, FSM models that over-approximate the actual invocation sequences.By contrast, our algorithms work on traces generated from multiple parts of the code and, potentially, from multiple applications.De Caso et al. [19] statically analyze C programs for invariants and use Contractor to create models that allow more behavior than the ground truth.

静态[19，53]和混合技术[17，58]为严格基于调用跟踪的规范挖掘提供了两种改变。肖汉等人，[53]从客户端代码推断出，有限状态机模型过于近似实际调用序列。相比之下，我们的算法工作在代码的多个部分生成的轨迹上，并且有可能来自多个应用程序。德·卡西欧等人，[19]静态分析不变量的程序，并使用承包商创建模型，允许比基本事实更多的行为。

ADABU [17] infers the concrete program state by statically find-ing side-effect-free invocations and combines that information with test case executions.While the concrete program state is also ab-stracted using predicates, these predicates are predetermined and do not relate multiple variables (e.g., ADABU abstracts integers only as negative, zero, or positive).Together with test case generation, ADABU can improve model quality, enabling code verification [16].However, this requires tailored unit test executions.Compared to TEMI, ADABU uses limited invariants and infers one model per runtime object, which hampers its applicability to rich classes and executions that involve many objects of the same type.Whaley et al. [58] create a separate submodel for each field of a class, an-alyzing if a method modifies a given field, and creating a 1-tail model that combines static and dynamic information about method invocations with respect to that field.Unlike TEMI, this approach requires static analysis, creates multiple submodels, and considers only one-step history (1-tail), limiting its applicability.

ADABU [17]通过静态查找无副作用调用来推断具体的程序状态，并将该信息与测试用例执行相结合。虽然具体的程序状态也是使用谓词来实现的，但是这些谓词是预先确定的，并且不涉及多个变量(例如，ADABU 只将整数抽象为负数、零或正数)。与测试用例生成一起，ADABU 可以提高模型质量，实现代码验证[16]。然而，这需要定制的单元测试执行。与 TEMI 相比，ADABU 使用有限的不变量，并推断每个运行时对象有一个模型，这妨碍了它对包含许多相同类型对象的丰富类和执行的适用性。Whaley 等人，[58]为一个类的每个字段创建一个单独的子模型，分析一个方法是否修改了一个给定的字段，并创建一个单尾模型，该模型结合了关于该字段的方法调用的静态和动态信息。与 TEMI 不同，这种方法需要静态分析，创建多个子模型，并且只考虑一步历史(单尾)，限制了它的适用性。

## 7.CONCLUSIONS

## 7.结论

Using a software library is a non-trivial task hampered by a lack of appropriate documentation for describing the required but often-implicit invocation protocols.This paper studied how different model-inference strategies perform when applied to libraries whose behavior is exercised using real software.The recent scalability improvements of the dynamic inference techniques have resolved many of the obstacles to their application in the real world: these techniques are now able to handle large sets of execution traces [4, 34] and large sets of runtime data values [45, 61].However, there are still noticeable gaps in understanding the quality of the models produced by model-inference techniques.

使用软件库是一项不平凡的任务，因为缺乏适当的文档来描述所需的但通常是隐式的调用协议。本文研究了不同的模型推理策略在应用于使用真实软件执行行为的库时是如何执行的。动态推理技术最近的可伸缩性改进解决了它们在现实世界中应用的许多障碍:这些技术现在能够处理大量的执行跟踪[4，34]和大量的运行时数据值[45，61]。然而，在理解模型推理技术产生的模型的质量方面仍然存在明显的差距。

As part of this research, we enhanced one existing technique, and presented two novel algorithms, SEKT and TEMI, that com-bine execution traces with automatically inferred program-state invariants.Our evaluation demonstrates that invariants-only and trace-enhanced-invariants significantly outperform the traces-only and invariant-enhanced-traces strategies.Trace-enhanced-invariants models, produced by TEMI, also exhibit superior recall, while being robust to noisy inputs.Our results highlight the significant impact of using program-state information to infer high-quality models.In addition, our research highlights the benefits of combining different types of runtime information to enhance inferred models and, in turn, to effectively support development tasks.In our future work, we plan to study whether this combination can enhance model-inference for concurrent libraries and multi-object protocols [34].

作为本研究的一部分，我们改进了现有的一种技术，并提出了两种新的算法，即 SEKT 和 TEMI 算法，这两种算法将执行轨迹与自动推断的程序状态不变量相结合。我们的评估表明，仅不变量和跟踪增强不

变量明显优于仅跟踪和不变增强跟踪策略。由 TEMI 公司生产的轨迹增强不变量模型也显示出优异的召回率，同时对噪声输入具有鲁棒性。我们的结果强调了使用程序状态信息推断高质量模型的重大影响。此外，我们的研究强调了组合不同类型的运行时信息的好处，以增强推断模型，进而有效地支持开发任务。在我们未来的工作中，我们计划研究这种组合是否能增强并发库和多对象协议的模型推理[34]。

## 8.ACKNOWLEDGMENTS

## 8.感谢

This work has been supported by the National Science Foundation under award numbers 1117593, 1218115, and 1321141.The work has also been supported in part by Infosys Technologies Ltd.

这项工作得到了国家科学基金会 1117593、1218115 和 1321141 的支持。这项工作也得到印孚瑟斯技术有限公司的部分支持



187

## 9.REFERENCES

## 9.参考

[1] N. Beckman, D. Kim, and J. Aldrich.An empirical study of object protocols in the wild.In the European Conference on Object-Oriented Programming (ECOOP), 2011.

[1]贝克曼、金和奥尔德里奇。野外目标协议的实证研究。在 2011 年欧洲面向对象编程会议上。

[2] I. Beschastnikh, Y. Brun, J. Abrahamson, M. D. Ernst, and A. Krishnamurthy.Unifying FSM-inference algorithms through declarative specification.In the International Confer-ence on Software Engineering (ICSE), 2013.

[2]贝斯恰斯提尼克、布伦、亚伯拉罕森、恩斯特和克里希那穆提。通过声明性规范统一 FSM 推理算法。软件工程国际会议(ICSE)，2013 年。

[3] I. Beschastnikh, Y. Brun, M. D. Ernst, and A. Krishnamurthy.Inferring Models of Concurrent Systems from Logs of their Behavior with CSight.In the International Conference on Software Engineering (ICSE), 2014.

[3]贝斯恰斯提尼克、布鲁恩、恩斯特和克里希那穆提。用 CSight 从并发系统的行为日志推断并发系统的模型。在 2014 年国际软件工程会议(ICSE)上。

[4] I. Beschastnikh, Y. Brun, S. Schneider, M. Sloan, and M. D. Ernst.Leveraging existing instrumentation to automatically infer invariant-constrained models.In the Joint Meeting of European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE), 2011.

[4]贝斯恰斯提尼克、布鲁恩、施耐德、斯隆和恩斯特。利用现有仪器自动推断不变约束模型。欧洲软件工程会议和软件工程基础研讨会联席会议(ESEC/FSE)，2011 年。

[5] A. Biermann and J. Feldman.On the synthesis of finite-state machines from samples of their behavior.IEEE Transactions on Computers, 21(6), 1972.

[5]比尔曼和费尔德曼。从有限状态机的行为样本合成有限状态机。IEEE 计算机交易，21(6)，1972。

[6] M. Bruch, M. Monperrus, and M. Mezini.Learning from exam-ples to improve code completion systems.In the Joint Meeting of European Software Engineering Conference and Sympo-sium on Foundations of Software Engineering (ESEC/FSE), 2009.

[6]布鲁赫先生、蒙皮鲁斯先生和梅奇尼先生。从考试中学习改进代码完成系统。在欧洲软件工程会议和软件工程基础研讨会的联席会议上(ESEC/FSE)，2009 年。

[7] R. P. Buse and W. Weimer.Synthesizing API usage exam-ples.In the International Conference on Software Engineering (ICSE), 2012.

[7]布思和魏默。合成应用编程接口使用测试。在 2012 年国际软件工程会议(ICSE)上。

[8] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith.Counterexample-guided Abstraction Refinement.In Computer Aided Verification, pages 154-169, 2000.

[8]克拉克、格鲁伯格、约翰·贾、吕英和维特。反例引导的抽象精化。《计算机辅助验证》，第 154-169 页，2000 年。

[9] Columba e-mail client.http://sourceforge.net/ projects/columba, 2013.

[9]哥伦布电子邮件客户端。http://sourceforge.net/项目/哥伦比亚大学，2013 年。

[10] J. Cook and A. Wolf.Discovering models of software pro-cesses from event-based data.ACM Transactions on Software Engineering and Methodology, 7(3), 1998.

[10]库克和狼。从基于事件的数据中发现软件过程的模型。软件工程和方法学的自动控制模块交易，7(3)，1998。

[11] C. Csallner, N. Tillmann, and Y. Smaragdakis.DySy: Dynamic symbolic execution for invariant inference.In the International Conference on Software Engineering (ICSE), 2008.

[11] C. Csallner，N. Tillmann 和 Y. Smaragdakis。DyY:用于不变推理的动态符号执行。在 2008 年国际软件工程会议(ICSE)上。

[12] DaCapo benchmark.http://www.dacapobench.org , 2009.

[12] DaCapo 基准。http://www.dacapobench.org，2009 年。

[13] B. Dagenais and M. Robillard.Creating and evolving devel-oper documentation: understanding the decisions of open source contributors.In the Symposium on Foundations of Soft-ware Engineering (FSE), 2010.

[[13]达吉那斯和罗比拉德。创建和发展开发文档:理解开源贡献者的决定。软件工程基础研讨会(FSE)，2010 年。

[14] B. Dagenais and M. Robillard.Recovering traceability links between an API and its learning resources.In the International Conference on Software Engineering (ICSE), 2012.

[14]达吉那斯和罗比拉德。恢复应用编程接口及其学习资源之间的可追溯性链接。在 2012 年国际软件工程会议(ICSE)上。

[15] The Daikon invariant detector.http://groups.csail.mit. edu/pag/daikon, 2009.

[15]戴康不变探测器。edu 麻省理工学院，2009 年。

[16] V. Dallmeier, N. Knopp, C. Mallon, G. Fraser, S. Hack, and A. Zeller.Automatically generating test cases for specification mining.IEEE Transactions on Software Engineering, 38(2), 2012.

[16]诉达尔梅尔、努普、马龙、弗雷泽、哈克和泽勒。为规范挖掘自动生成测试用例。IEEE 软件工程交易，38(2)，2012。

[17] V. Dallmeier, C. Lindig, A. Wasylkowski, and A. Zeller.Min-ing object behavior with ADABU.In the Workshop on Dy-namic Analysis (WODA), 2006.

[17]诉达尔梅尔、林迪希、瓦西里科夫斯基和泽勒。用 ADABU 最小化对象行为。动力学分析研讨会(达沃)，2006 年。

[18] G. de Caso, V. Braberman, D. Garbervetsky, and S. Uchitel.Automated abstractions for contract validation.IEEE Transac-tions on Software Engineering, 38(1), 2012.

[18] G. de Caso，V. Braberman，D. Garbervetsky 和 S. Uchitel。合同验证的自动化抽象。IEEE 软件工程交易，38(1)，2012。

[19] G. de Caso, V. Braberman, D. Garbervetsky, and S. Uchitel.Enabledness-based program abstractions for behavior valida-tion.ACM Transactions on Software Engineering and Method-ology, 22(3), 2013.

[19] G. de Caso，V. Braberman，D. Garbervetsky 和 S. Uchitel。基于使能性的行为验证程序抽象。《软件工程和方法学的自动控制管理交易》，22(3)，2013 年。

[20] M. D. Ernst, J. H. Perkins, P. J. Guo, S. McCamant, C. Pacheco, M. S. Tschantz, and C. Xiao.The Daikon system for dynamic detection of likely invariants.Science of Computer Program-ming, 69(1), 2007.

[20]恩斯特、珀金斯、郭炳江、麦卡曼、帕切科、茨尚茨和肖振国。动态检测可能不变量的 Daikon 系统。计算机程序科学——明，69(1)，2007。

[21] M. Gabel and Z. Su.Javert: Fully automatic mining of general temporal properties from dynamic traces.In the Symposium on Foundations of Software Engineering (FSE), 2008.

[21]贾布尔和苏志坚。沙威:从动态轨迹中完全自动挖掘一般时间属性。软件工程基础研讨会(FSE)，2008 年。

[22] M. Gabel and Z. Su.Online inference and enforcement of tem-poral properties.In the International Conference on Software Engineering (ICSE), 2010.

[22]贾布尔和苏志坚。暂时性质的在线推断和实施。在 2010 年国际软件工程会议(ICSE)上。

[23] D. Garlan, R. Allen, and J. Ockerbloom.Architectural mis-match: Why reuse is still so hard.IEEE Software, 26(4), 2009.

[23]贾兰、艾伦和奥克布勒。架构不匹配:为什么重用仍然如此困难。IEEE 软件，26(4)，2009。

[24] C. Ghezzi, M. Pezzè, M. Sama, and G. Tamburrelli.Mining Behavior Models from User-intensive Web Applications.In the International Conference on Software Engineering (ICSE), 2014.

[24]盖兹、佩兹、萨马和坦伯利。从用户密集型网络应用程序中挖掘行为模型。在 2014 年国际软件工程会议(ICSE)上。

[25] JarInstaller.http://sourceforge.net/projects/ kurumix, 2013.

[25] JarInstaller。http://sourceforge.net/projects/·库鲁米克斯，2013 年。

[26] jEdit.http://www.jedit.org , 2014.

[26]杰迪。http://www.jedit.org，2014 年。

[27] JFtp client.http://j-ftp.sourceforge.net , 2013.

[27]JTp 客户端。http://j-ftp.sourceforge.net，2013 年。

[28] jlGUI.http://www.javazoom.net/jlgui/jlgui.html, 2010.

[28] jlGUI。http://www.javazoom.net/jlgui/jlgui.html, 2010。

[29] I. Krka, Y. Brun, G. Edwards, and N. Medvidovic.Synthe-sizing partial component-level behavior models from system specifications.In the Joint Meeting of European Software Engineering Conference and Symposium on Foundations of Soft-ware Engineering (ESEC/FSE), 2009.

[29]伊·克尔卡、伊·布伦、爱德华兹和梅德维多维奇。根据系统规范合成部分组件级行为模型。2009 年欧洲软件工程会议暨软件工程基础研讨会联席会议(ESEC/FSE)。

[30] I. Krka, Y. Brun, and N. Medvidovic.Automatically min-ing specifications from invocation traces and method invari-ants.Technical Report CSSE-2013-509, Center for Systems and Software Engineering, University of Southern California, 2013.

[30]伊·克尔卡、伊·布伦和恩·梅德维多维奇。从调用跟踪和方法变量中自动最小化规范。CSSE-2013-509 技术报告，南加州大学系统和软件工程中心，2013 年。

[31] I. Krka, Y. Brun, D. Popescu, J. Garcia, and N. Medvidovic.Using dynamic execution traces and program invariants to enhance behavioral model inference.In the International Con-ference on Software Engineering New Ideas and Emerging Results Track (ICSE NIER), 2010.

[31]伊·克尔卡、伊·布伦、德·波佩斯库、加西亚和梅德维多维奇。使用动态执行跟踪和程序不变量来增强行为模型推理。在 2010 年软件工程新理念和新成果跟踪国际会议(ICSE NIER)上。

[32] S. Kumar, S.-C. Khoo, A. Roychoudhury, and D. Lo.Inferring class level specifications for distributed systems.In the Inter-national Conference on Software Engineering (ICSE), 2012.

[32]库马尔、霍奥、罗伊乔杜里和劳。推断分布式系统的类级规范。软件工程国际会议(ICSE)，2012 年。

[33] K. G. Larsen and B. Thomsen.A modal process logic.Logic in Computer Science, 1988.

[33]拉森和汤姆森。模态过程逻辑。计算机科学中的逻辑，1988。

[34] C. Lee, F. Chen, and G. Ro su.Mining parametric spec-ifications.In the International Conference on Software Engineering (ICSE), 2011.

[34]李春江，陈福林和苏广荣。挖掘参数规范。在 2011 年国际软件工程会议(ICSE)上。

[35] K. Li, C. Reichenbach, Y. Smaragdakis, and M. Young.Second-order constraints in dynamic invariant inference.In the Joint Meeting of European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE), 2013.

[35]李开复、赖兴巴赫、斯马拉格达基斯和杨梅杰。动态不变推理中的二阶约束。欧洲软件工程会议和软件工程基础研讨会联席会议(ESEC/FSE)，2013 年。

[36] D. Lo and S. Khoo.QUARK: Empirical assessment of automaton-based specification miners.In the Working Conference on Reverse Engineering (WCRE), 2006.

[36]劳和霍。夸克:基于自动机的规范挖掘器的经验评估。逆向工程工作会议(WCRE)，2006 年。

[37] D. Lo and S. Khoo.SMArTIC: Towards building an accurate, robust and scalable specification miner.In the Symposium on Foundations of Software Engineering (FSE), 2006.

[37]劳和霍。智能集成电路:致力于构建一个精确、健壮和可扩展的规范挖掘器。软件工程基础研讨会(FSE)，2006 年。



188

[38] D. Lo and S. Maoz.Scenario-based and value-based specifica-tion mining: Better together.In the International Conference on Automated Software Engineering (ICSE), 2010.

[38]劳和毛兹。基于场景和基于价值的特定挖掘:更好地结合在一起。自动化软件工程国际会议(ICSE)，2010 年。

[39] D. Lo, L. Mariani, and M. Pezzè.Automatic steering of behavioral model inference.In the Joint Meeting of European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE), 2009.

[39]洛城、马里亚尼和佩兹。行为模型推理的自动导向。欧洲软件工程会议和软件工程基础研讨会联席会议(ESEC/FSE)，2009 年。

[40] D. Lo, L. Mariani, and M. Santoro.Learning extended fsa from software: An empirical assessment.Journal of Systems and Software, 85(9), 2012.

[40]洛城，马里亚尼和桑托罗。从软件中学习扩展 fsa:一个经验评估。系统和软件杂志，85(9)，2012。

[41] D. Lorenzoli, L. Mariani, and M. Pezzè.Automatic generation of software behavioral models.In the International Conference on Software Engineering (ICSE), 2008.

[41]洛伦佐利、马里亚尼和佩兹。软件行为模型的自动生成。在 2008 年国际软件工程会议(ICSE)上。

[42] K. Mu slu, Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin.Speculative analysis of integrated development environment recommendations.In the Conference on Object-Oriented Pro-gramming, Systems, Languages and Applications (OOPSLA), 2012.

[42]克·穆·斯卢、伊·布伦、福尔摩斯、恩斯特和诺特金。综合发展环境建议的推测性分析。在 2012 年面向对象程序设计、系统、语言和应用会议上。

[43] T. Ohmann, M. Herzberg, S. Fiss, A. Halbert, M. Palyart, I. Beschastnikh, and Y. Brun.Behavioral Resource-Aware Model Inference.In International Conference On Automated Software Engineering (ASE), Västerås, Sweden, 2014.

[43]奥曼、赫尔茨贝格、菲丝、哈尔伯特、波莱特、贝沙斯蒂尼克和布伦。行为资源感知模型推理。自动化软件工程国际会议，瑞典维尔斯特拉斯，2014 年。

[44] T. Ohmann, K. Thai, I. Beschastnikh, and Y. Brun.Mining Precise Performance-Aware Behavioral Models from Existing Instrumentation.In the International Conference on Software Engineering New Ideas and Emerging Results (ICSE NIER) track, 2014.

[44]奥曼、泰拳、贝斯恰斯蒂尼克和布鲁恩。从现有仪器中挖掘精确的性能感知行为模型。在 2014 年软件工程新理念和新成果国际会议(ICSE NIER)上。

[45] J. H. Perkins, S. Kim, S. Larsen, S. Amarasinghe, J. Bachrach, M. Carbin, C. Pacheco, F. Sherwood, S. Sidiroglou, G. Sulli-van, et al. Automatically patching errors in deployed software.In the Symposium on Operating Systems Principles (SOSP), 2009.

[45] J. H .珀金斯、s .金、s .拉森、s .阿马拉辛格、j .巴赫拉赫、m .卡宾、c .帕切科、f .舍伍德、s .西迪罗格洛、g .苏利-范等。自动修补已部署软件中的错误。在操作系统原则研讨会(SOSP)上，2009 年。

[46] N. Polikarpova, I. Ciupa, and B. Meyer.A comparative study of programmer-written and automatically inferred contracts.In the International Symposium on Software Testing and Analysis (ISSTA), 2009.

[46]波利卡波娃、伊·西帕和迈耶。程序员编写的合同和自动推断的合同的比较研究。在 2009 年国际软件测试与分析研讨会上。

[47] M. Pradel, P. Bichsel, and T. R. Gross.A framework for the evaluation of specification miners based on finite state machines.In the International Conference on Software Maintenance (ICSM), 2010.

[47]普拉德、比谢尔和格罗斯。基于有限状态机的规范挖掘器评估框架。在 2010 年国际软件维护会议(ICSM)上。

[48] M. Pradel and T. R. Gross.Leveraging test generation and

[48]普拉德勒和格罗斯。利用测试生成和

specification mining for automated bug detection without false positives.In the International Conference on Software Engineering (ICSE), 2012.

没有误报的自动错误检测的规范挖掘。在 2012 年国际软件工程会议(ICSE)上。

[49] S. P. Reiss and M. Renieris.Encoding program executions.In the International Conference on Software Engineering (ICSE), 2001.

[49]莱斯和雷尼尔斯。编码程序执行。在软件工程国际会议(ICSE)上，2001 年。

[50] R. Robbes and M. Lanza.How program history can im-prove code completion.In the International Conference on Automated Software Engineering (ASE), 2008.

[50]罗柏和兰扎。程序历史如何改进代码完成。在 2008 年国际自动化软件工程会议上。

[51] M. Robillard.What makes APIs hard to learn?Answers from developers.IEEE Software, 26(6), 2009.

[51]罗比拉德。什么让 APIs 很难学习？开发者的答案。IEEE 软件，26(6)，2009。

[52] M. Schur, A. Roth, and A. Zeller.Mining behavior models from enterprise web applications.In the Joint Meeting of European Software Engineering Conference and Symposium on Foundations of Software Engineering (ESEC/FSE), 2013.

[52]舒尔、罗斯和泽勒。从企业网络应用程序中挖掘行为模型。欧洲软件工程会议和软件工程基础研讨会联席会议(ESEC/FSE)，2013 年。

[53] S. Shoham, E. Yahav, S. J. Fink, and M. Pistoia.Static Specification Mining Using Automata-Based Abstractions.IEEE Transactions on Software Engineering, 34(5), 2008.

[53]绍姆、叶海夫、芬克和皮斯托亚。使用基于自动机的抽象进行静态规范挖掘。IEEE 软件工程交易，34(5)，2008。

[54] R. N. Taylor, N. Medvidovic, and E. M. Dashofy.Software Architecture: Foundations, Theory, and Practice.John Wiley & Sons, 2009.

[54]泰勒、梅德维多维奇和达索耶。软件架构:基础、理论和实践。约翰·威利父子公司，2009。

[55] Project Voldemort.http://www.project-voldemort.com , 2014.

[55]伏地魔计划。http://www.project-voldemort.com，2014 年。

[56] N. Walkinshaw and K. Bogdanov.Inferring finite-state models with temporal constraints.In the International Conference on Automated Software Engineering (ASE), 2008.

[56]沃金肖和鲍格丹诺夫。推断具有时间约束的有限状态模型。在 2008 年国际自动化软件工程会议上。

[57] Y. Wei, C. A. Furia, N. Kazmin, and B. Meyer.Inferring better contracts.In the International Conference on Software Engineering (ICSE), 2011.

[57]魏英伟、福里亚、卡兹明和迈耶。推断出更好的合同。在 2011 年国际软件工程会议(ICSE)上。

[58] J. Whaley, M. C. Martin, and M. S. Lam.Automatic extraction of object-oriented component interfaces.In the International Symposium on Software Testing and Analysis (ISSTA), 2002.

[58] J. Whaley，M. C. Martin 和 M. S. Lam。面向对象组件接口的自动提取。软件测试与分析国际研讨会，2002 年。

[59] T. Xie et al. Data mining for software engineering.Computer, 42(8), 2009.

[59]谢天华等.软件工程中的数据挖掘。计算机，42(8)，2009。

[60] J. Yang, D. Evans, D. Bhardwaj, T. Bhat, and M. Das.Perra-cotta: Mining temporal API rules from imperfect traces.In the International Conference on Software Engineering, 2006.

[60]杨杰，埃文斯，巴德瓦杰，巴特和达斯。从不完美的轨迹中挖掘时态应用编程接口规则。在 2006 年国际软件工程会议上。

[61] Yices SMT Solver.http://yices.csl.sri.com , 2009.

[61]易斯 SMT 求解器。http://yices.csl.sri.com，2009 年。

[62] S. Zhang, D. Saff, Y. Bu, and M. D. Ernst.Combined static and dynamic automated test generation.In the International Symposium on Software Testing and Analysis (ISSTA), 2011.

[62]张士元、萨夫、布和恩斯特。静态和动态联合自动测试生成。在 2011 年国际软件测试与分析研讨会上。



189