参数化运行时监控研究

王 珍 叶俊民 陈 曙 辜 剑 金 聪
(华中师范大学计算机学院 武汉 430079)

摘 要 随着计算机软件广泛应用于各类安全关键系统以及软件日趋复杂,软件可靠性变得越来越重要。作为一种广泛使用于各种平台的软件解决方案,运行时监控是提高软件可靠性的最灵活的解决方案之一。但随着运行时监控技术以及软件技术的发展,人们希望通过运行时监控技术来验证系统的动态属性,从而提出参数化性质的运行时监控技术。由于其在面向对象系统中的适用性,参数化性质的运行时监控已经受到了越来越多的关注。综述了参数化运行时监控的研究进展,提出了参数化运行时监控的问题定义,介绍了这一领域的主要研究内容:参数化运行时监控方法、减少参数化监控开销的技术、多属性规约的参数化运行时监控。

关键词 运行时监控,参数化性质规约,参数化运行时监控
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2014.11.029

Research on Parameterized Runtime Monitoring

WANG Zhen YE Jun-min CHEN Shu GU Jian JIN Cong
(School of Computer Science, Central China Normal University, Wuhan 430079, China)

Abstract With the wide application of software in all kinds of safety critical systems as well as the increasingly complexity, software reliability becomes more and more important. As a software solution widely used in various platforms, runtime monitoring is one of the most flexible solution to enhance the reliability of software. With the development of runtime monitoring and software technology, people want to verify the dynamic properties of system through runtime monitoring. So runtime monitoring of parametric properties was presented. Runtime monitoring of parametric properties have achieved more and more attention because of its applicability in the object-oriented system. This paper summarized the researches on parametric runtime verification, presented the problem definition of parametric runtime verification, and introduced the main research content of this field, including parametric runtime verification approaches, technologies of reducing parametric monitoring overhead and runtime monitoring of multiply parametric properties.

Keywords Runtime monitoring, Parametric property specification, Parametric runtime monitoring

1 引言

随着计算机软件广泛应用于各类安全关键系统以及软件日趋复杂,软件失效经常给生命、经济和国家安全等方面造成不可挽回的巨大损失,因此研究如何提高软件质量成为当前的一个研究热点。

软件测试是一种常用的提高软件质量的方法,但它只能发现系统的错误,不能确保系统没有错。且随着软件日益复杂,软件测试技术已不能满足人们的需求。之后,研究学者提出使用形式化方法[1]。其主要包括以模型检验为代表的静态验证技术[2]和以定理证明为代表的演绎推理技术[3]。但定理证明过程需要人工干预,自动化程度低;模型检验技术针对的是软件系统模型,而不是实际的运行,且当系统较复杂时,容易产生"状态爆炸"问题。总之,传统的验证技术不能保证部署后的软件系统在实际运行过程中安全和可靠。

为解决上述问题,人们提出运行时监控技术,它是通过动态监控和分析系统的执行来看其是否满足指定的性质。Blum 和 Kannan 最早提出在运行时检验程序的正确性[4],关于程序检查的论文证实运行时监控相比静态验证更加灵活,这使得更多学者研究与运行时监控相关的问题,到目前为止,研究学者已提出了许多高效的在线检查算法[5-7],且已用于各种工具中[8-10]。然而,随着运行时监控技术以及软件技术的发展,人们希望通过运行时监控技术来验证更为复杂的系统属性,比如对于程序中普遍存在的动态属性或那些很难通过静态技术精确验证的性质,这类需求往往要求特定类型的所有对象或数据结构的所有实例都满足或不满足某个特定的属

到稿日期:2013-09-21 返修日期:2013-11-20 本文受湖北省自然科学基金面向项目(2010CDB04001),武汉大学计算机软件工程国家重点实验室开放基金项目(SKLSE 20080705),华中师范大学基本科研业务基金项目(CCNU11A02007),华中师范大学自制实验仪器设备与软件项目(201314)资助。

王 珍(1989—),女,硕士生,主要研究方向为软件工程,E-mail:578540211@qq.com;叶俊民(1965—),男,博士,教授,主要研究方向为软件可靠性,E-mail:jmye@mail.ccnu.edu.cn;陈 曙(1981—),男,博士,讲师,主要研究方向软件工程、软件形式化;辜 剑(1987—),男,硕士生,主要研究方向为软件工程;金 聪(1960—),女,博士,教授,主要研究方向为数字水印、模式识别。

性。如,在面向对象程序中,经常会要求其动态实例化后产生的所有对象都满足某个特定时序性质。不难发现,仅采用一般的运行时监控技术已经不能满足需要。这就需要在性质规约中引入参数,对参数化性质进行监控的技术称为参数化运行时监控。

本文首先给出参数化监控的问题定义,接下来介绍参数化监控规约和监控方法,然后介绍常用的减少参数化监控开销的方法,最后对未来研究进行展望。

2 问题定义

随着运行时监控技术的发展及软件日趋复杂,人们希望验证与软件动态特征相关的性质,如变量赋值、存储空间的动态分配与释放、新进程或线程的启动等。对于这类性质,若仅仅采用命题符号来描述原子命题已经不能满足需求,需要引入参数化命题来描述这些与软件动态特征相关的性质。

参数化性质是包含自由变量且用于描述对象行为的性质,参数在运行时绑定到具体的数据或对象,参数化运行时监控技术主要用于面向对象程序中,它通过动态监控和分析面向对象程序的执行,检查其是否满足指定的参数化性质。与非参数化运行时监控技术相比,它具有以下特征:

· 参数化性质通常是描述程序在执行过程中应遵循的对象行为的性质,而非参数化性质只能描述系统全局行为。

· 在参数化性质的运行时监控中,需要考虑变量的赋值集合的提取和绑定操作,即变量赋值集合的获取以及实例化过程,经过多次迭代,可能形成多个完全实例化的被监控性质,这些实例化性质的运行时验证始于不同的状态且可并发进行。

· 在参数化性质的运行时监控中,进行变量赋值和绑定操作,同时监控多个完全实例化的性质。这些操作所产生的运行时开销可能会影响系统性能。

对于参数化性质的形式化表示,通常是通过引入参数化命题和量词∀、∃,扩展一般的形式逻辑规约来描述参数化性质。如文献[11]提出 P_ALTL 来表示参数化性质,通过引入参数化命题和存在谓词扩展 LTL,来支持参数化性质的形式化表示。文献[12]提出使用 DLTL 来描述参数化性质,它是利用 AspectJ 具有动态绑定自由变量的特征来支持参数化的性质描述,但它只支持有限数量的参数。

由于参数化性质是包含参数的,因而其分析对象是在程序运行过程中动态产生的,这使得其监控方法与一般的监控方法有所不同:首先是监控过程中需进行变量赋值集合的提取和绑定操作;其次是,经过变量赋值集合提取和绑定操作的多次迭代后,可形成多个完全实例化的性质。因此参数化运行时监控方法是参数化运行时监控的一个重要研究内容。

参数化运行时监控的另一研究内容是管理运行时开销。运行时开销主要有两个来源。其一是从一个运行系统提取观测信息的插装操作,虽然减少插装点的数量可以减少插装开销,但也可能导致遗漏观测信息和检查结果不正确。监控器的管理是开销的另一个主要来源,减少不必要监控器实例的个数以及采用有效的监控器查找算法都可以减少运行时监控开销,同时采用有效的监控器回收算法可以减少存储开销。

下文将分别介绍参数化性质规约、参数化监控方法以及如何减少运行时参数化监控开销。

3 参数化性质规约

参数化性质是包含自由变量且用于描述对象行为的性质,参数在运行时绑定到具体的数据或对象上,参数化性质不仅可以描述全局行为,也可以描述对象的行为,参数化性质可以通过多种不同的规约形式来描述,目前主要有以下几种形式化方法。

1. 基于时序逻辑的规约形式

基于时序逻辑的规约形式在模型检查中被证明是描述反应式系统时序性质的有效手段,因此被广泛应用于运行时监控研究中。如文献[11]提出 P_ALTL 来表示参数化性质,它通过引入参数化命题和存在谓词扩展 LTL,如 $\varphi=\forall x \forall y: p(x,y) \rightarrow q(x,y)$,其中 $p(x,y)$、$q(x,y)$为参数化命题,x、y 为自由变量,且都被量词∀修饰。

文献[12]提出 DLTL,它利用 AspectJ 具有动态绑定自由变量的特征来支持参数化性质描述,由于 AspectJ 使我们可在每个连接点的前面和后面执行一段代码,利用这个特征,将连接点的入口事件和出口事件定义为原子命题,命题就可以定义和访问运行绑定到对象上的变量,从而 DLTL 就可以描述参数化性质。如图 1 所示为文献[12]中给出的一个简单的 DLTL 性质公式。公式的开始声明了一个类型为 Stack 的变量 s,该实例中,第 3 行指定了调用一个创建栈的方法的出口事件,且绑定变量 s,第 7 行指定了调用栈 s 的 push 方法的入口事件。该性质公式表示无论什么时候创建了一个新的栈 s,最终都会调用栈 s 的 push 函数。由于性质公式的命题中绑定了相应的变量,且命题是连接点的入口事件和出口事件,因此,在系统运行过程中,可以通过这些入口出口事件访问绑定到对象上的变量。

```
1. Stack s;
2. G( (
3. exit(call(stack. new(. .))) returning s
4. ) -> (
5. X(
6. F(
7. entry(call( * Stack. push(Object))&&target(s))
8. ) ) ) )
```

图 1 一个 DLTL 表示的栈实例

2. 面向语言的规约形式

面向语言的形式化方法能有效地利用相关语言的一些特征,来简化运行时监控过程。如文献[13]中使用扩展的正则表达式来表示验证的性质,在运行时监控过程中,其将正则表达式转换成一个有限自动机,用变量绑定来标记相应的状态。文献[13]中给出 *CREATE NEXT * UPDATE+ NEXT*,其为正则表达式表示的一个性质。其主要包含两个变量 e 和 v,*CREATE* 表示从迭代器 v 创建一个枚举对象 e,*UPDATE* 表示修改 v,*NEXT* 表示调用 e 上的方法 *nextElement*()。将该正则表达式转换成一个有限自动机,然后对于接收的事件,将其变量绑定作为约束来标记后续状态。当终止状态上的标记为空时,则表示不存在满足该正则表达式的轨迹切片。否则其标记的约束中对象的对应轨迹切片与正则表达式匹配。如一个轨迹为 $CREATE(v_1,e_1)$ $CREATE(v_1,e_2)$ $NEXT(e_2)$ $NEXT(e_1)$ $UPDATE(v_1)$ $NEXT(e_1)$,得到的标记自动机如

图 2 所示，则可知对应于 $v=v_1 \wedge e=e_1$ 的轨迹切片 $CREATE(v_1,e_1)\ NEXT(e_1)\ UPDATE(v_1)\ NEXT(e_1)$ 与正则表达式匹配。

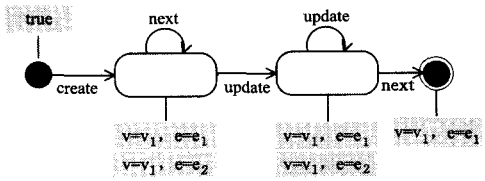

图 2　状态标记自动机

3. 面向查询的语言

使用面向查询的语言可以描述一些比较特殊的性质，如 PQL[15] 是一种基于上下文无关文法的语言，它可表示一大类特定应用程序的代码模式，且支持无穷数量的参数。如图 3 所示为一个简单的 PQL 表示的 SQL 注入查询的例子[15]。开始定义查询的名称，uses 声明使用的变量，matches 部分指定匹配的模式，最后指定发生匹配时所执行的动作。对于该实例，静态检查器报告所有这些情况，即调用 r 对象的 getParameter 方法所返回的对象 p 是 c 对象的 execute 方法的一个参数。动态检查器则在运行时发现这些匹配，若发现匹配，则阻止 execute 方法的调用，而执行 Util. CheckedSQL。

PTQL[16] 是一种类 SQL 关系查询的语言，结构与 SQL 查询类似，程序员可使用该语言表示程序行为的查询，通过在查询中引入参数支持参数化性质的验证。SQL 注入查询如图 3 所示。

```
query simpleSQLInjection ()
uses
  object HttpServletRequest r;
  object Connection c;
  object String p;
matches { p=r. getParameter(_); }
replaces c. execute(p)
  with Util. CheckedSQL(c,p)
```

图 3　SQL 注入查询

4. 基于规则的方法

目前研究的基于规则的方法有 Eagle[17] 和 RlueR[18]，它们可用来定义和实现多种有穷轨迹监控逻辑，包括未来和过去时间时序逻辑、扩展的正则表达式、区间逻辑等，它们支持递归的参数化方程，通过规则定义来定义新的操作符，规则定义以公式和数值作为参数，从而可以被用来定义和实现支持参数化属性逻辑。如文献[17]中的一个简单以数值作为参数的实例，一个量化 LTL 性质公式 $\Box(x>0\rightarrow \exists k.(x=k \wedge \Diamond y=k))$，表示在某个时刻 $x=k$ 且 $k>0$ 时，最终 y 也等于 k。该性质可以表示成，$\underline{\min} R(\underline{\text{int}}\ k)=Eventually(y=k)$，$\underline{mon}\ M=Always(x>0\rightarrow R(x))$，规则 R 中以整数 k 作为参数，k 在所定义的监控器 M 中进行实例化。

在上述参数化性质规约中，tracematch[14] 和 DLTL[12] 只支持有限数量的参数化性质表示。其它规约形式只支持无限数量的参数。

4　参数化运行时监控方法

参数化运行时监控是非参数化运行时监控的扩展，由于参数化规约的分析对象是在程序运行过程中动态产生的，而且不同运行涉及的对象数目也可能完全不同，因此很难预先静态确定分析对象的值域，需要在程序运行过程中动态地确定所有变量的当前取值，实例化的被监控性质的运行时监控过程对应于一般非参数化性质的运行时监控过程。

目前为止，已有很多有效的参数化运行时监控方法，根据变量绑定与性质检查间的关系将参数化性质监控方法分为两大类[31]。第一类称为单一整体监控（集中式监控）方法，在该类参数化性质监控中，参数绑定的处理和性质检查是紧密关联的。对于每一个参数化性质，生成一个参数化监控器，该监控器以特定逻辑形式的方法接收和处理每个参数化事件。第二类称为分散式监控，在该类监控方法中，分别处理参数绑定与性质检查。对于一个参数化性质，生成多个监控器实例，每个监控器实例对应于一个参数实例，每个参数实例的运行时监控过程等同于一个非参数化性质的运行时监控过程。

4.1　单一整体监控

单一整体监控，即由参数化性质生成一个单一的参数化监控器，用该监控器监控整个程序的运行，处理程序中出现的所有参数化事件。该方法的监控原理图如图 4 所示。

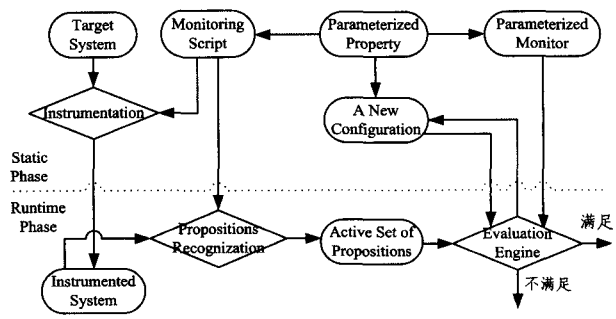

图 4　单一整体监控原理图

该监控过程可分为两个阶段：(1)静态阶段，即目标程序执行之前，根据参数化性质规约，自动生成参数化监控器和相关监控脚本，并将监控代码插装到目标程序中。(2)运行时阶段，在该阶段中，提取系统运行时相关信息，完成变量赋值集合的提取与绑定，检查属性规约是否被满足，如果检测到性质背离，则给出性质背离信息，下面具体说明。

(1)静态阶段

监控代码的插装：该过程与非参数化运行时监控相同，目前最普遍的方式是使用面向方面编程来实现源代码的插装，J-LO[12] 中率先将面向方面技术应用于运行时监控中。然而 tracematches[14] 体现了面向方面插装与运行时监控技术之间更加紧密的关系。tracematches 是 AspectJ 语言的一种扩展，能捕获事件的正则模式和表示 aspect 语义，从而可以通过编译器将其自动植入程序中，因此，一些运行时监控工具，如 MOP[25] 依赖于 AspectJ 完成代码插装。

参数化监控器的构造：参数化监控器的构造也是基于自动机理论，通常将性质公式转化为交错 Buch 自动机，与一般的非参数化的自动机不同的是，前者中每个状态都为抽象状态，在系统执行过程中，抽象状态将绑定不同的变量赋值，从而对应于不同的具体状态。对于性质公式 φ，先求出其闭包 $cl(\varphi)$，其对应的参数化自动机 $A(\varphi)=\langle \Sigma,Q,q_0,\delta,F\rangle$，由于每个状态的后继状态可以为无穷多个，因此引入一个 split 操作，把公式分解为两个部分，第一部分是为了公式 φ 最终被满足，必须在当前状态处被满足的那部分子公式，第二部分为需

要在未来某个时刻被满足的部分子公式。对于性质公式 φ，先求出其闭包 $cl(\varphi)$，则 $cl(\varphi)$ 即为公式 φ 对应的参数化交错自动机的抽象状态集合，其初始状态为 $q_0=\varphi$，迁移关系通过 split(Q)得到，其得到结果的第一部分为迁移边上标记的公式，第二部分即为迁移边指向的状态。其具体分解和转化过程可参考文献[11]。

(2)运行时阶段

由于监控器的每个抽象状态都可能对应于许多不同的具体状态，因此，创建完整的对应于不同的具体状态的自动机是比较复杂的，通常采用 on-the-fly 的方式把具体的变量赋值与抽象的监控器状态关联起来，同时判断抽象迁移的使能情况。监控过程中，从初始状态出发，识别当前状态所满足的命题集，基于该状态和其满足的命题集提取变量赋值集合，根据参数化监控器确定迁移的使能情况，迁移发生时，绑定变量赋值，产生一个新的格局，其中包含与监控器抽象状态关联的具体状态，直到性质公式被完全实例化后，判断性质公式是否被满足或是否最终到达一个可接受状态。如 J-LO[12] 中使用 DLTL 来描述其待验证的性质规约，然后借助于交错自动机实现监控，将待验证的性质公式作为自动机的初始格局，然后根据变量赋值提取与绑定生成其后继格局，直到执行轨迹终止。若最终生成的格局是属于自动机的可接受状态集，则表明系统执行满足性质规约，否则发生性质背离。

目前大部分的参数化运行时监控系统如文献[12,14-18]都是采用这种监控方式，但这种监控方式面临的最大挑战是：在性质检查过程中，如何跟踪每个特定参数实例的状态，当监控器的某些部分变得无关紧要时，如何对这些部分进行垃圾回收。且这种将变量绑定与性质检查紧密关联的监控方法会导致监控算法更加复杂并只能针对特定性质规约。

4.2 分散式监控

分散式监控与单一整体监控不同，它将变量绑定与性质检查分开处理，在监控过程中，生成多个监控器，每个监控器对应一个参数实例。在性质检查过程中跟踪该参数实例的状态。这一类监控方法的具体监控原理图如图 5 所示。

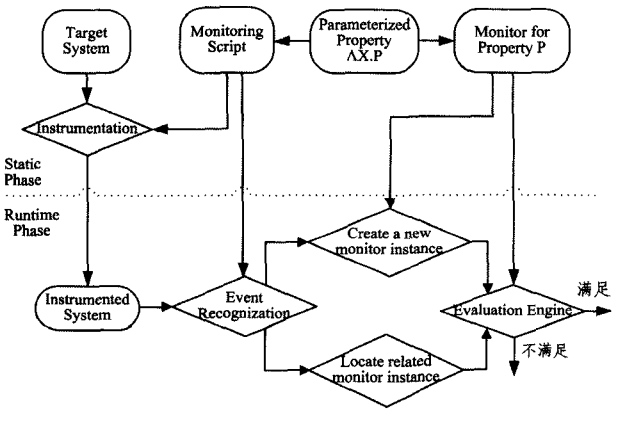

图 5 分散式监控原理图

与单一整体监控方法类似，该监控方法也可分为静态和运行时两个阶段，与单一整体监控方法不同的是，在静态阶段，该方法是构造参数化性质公式 $\Lambda X.P$ 中公式 P 的监控器，而不是构造参数化性质公式 $\Lambda X.P$ 的监控器。且该过程构造的监控器与一般监控器有所不同，对于性质 P 的监控器 M 为($S,\varepsilon,C,\iota,\sigma:S\times\varepsilon\to S,\gamma:S\to C$)，它包含一个输出类别集 C 和一个输出函数 γ，C 可包含 validating、violating 和 don't know 3 个元素。而输出函数 γ 用于判断截止到当前状态，所发生的事件轨迹属于哪个类别。

在运行时阶段，当接收到一个参数化事件 e 后，使用轨迹切片算法[28,30,31]将其分配到对应的轨迹切片，然后判断是否出现了新的参数实例，若是，则创建对应于该参数实例的监控器实例，否则根据参数实例检索相关的监控器实例，并更新监控器实例状态。该过程是动态进行的，不需要事先生成所有的监控器状态，而是根据需要生成相应的监控器实例状态，每次生成下一状态后，删除前一状态，这样可节省存储开销，监控开始时生成一个空参数实例的监控器实例。在更新了监控器实例状态得到一个新的状态 s 后，检查其输出 $\gamma(s)$ 是否违背了性质，若是，则报告错误信息给用户。

这种监控方式产生的运行开销较第一种方式低，由于变量绑定和性质检查是分开进行处理的，其能应用多种优化方法，且适合于各种不同的性质规约形式。但目前该方法只适用于最外层参数化的属性，即不考虑嵌套参数。

5 减少参数化监控开销方法

监控包含的许多系统变量的复杂性质会给系统造成很大的监控开销，从而影响系统性能，在参数化运行时监控中，监控开销取决于许多因素，包括需观察的程序位置个数、不同的数据值需要监控器多个副本的程度，以及更新监控器状态和检查性质违背的开销等。目前主要用于减少参数化监控开销的方法有：程序静态分析，以及针对分散式监控方法的索引机制、enable 集优化、监控器垃圾回收等。

5.1 程序静态分析

静态分析与动态分析相结合的概念源于编程语言中的 typestate 分析环境中，而静态 typestate 分析往往会导致不确定的结果，但它会产生一些有价值的信息，这些信息可以简化运行时进行分析的状态模型。因此，研究学者在此基础上提出了残留 typestate 分析[19]。其后，Eric Bodden 等人[20]提出通过快速检查和流不敏感基于指针分析来发现和删除不必要的监控插装，但该方法只限于 tracematches。文献[21]提出一种新的 AspectJ 语言的扩展，其依赖通知且使用上述方法的本质，进行三阶段的分析旨在确定那些不相关的程序点，通过修改这些程序点的动态残留为永远不匹配来禁用这些程序点，减少运行时监控的插装点数量，从而减少运行时监控开销。该方法可应用于基于历史的方面，且依赖通知使得优化独立于所选择的代码生成工具和规约形式。该工作的结果已在工具 Clara[22] 中实现。本质上，这些优化技术更像是编译器优化。它们首先执行静态分析得到程序和性质之间如何进行关联的信息。然后，使用那些分析的结果删除或修改程序中与监控相关的插装点，从而减少运行时开销。

其后，Avgustinov 等人[13]提出一种对运行时监控本身进行优化的静态分析方法，首先使用弱引用的概念，删除那些被垃圾回收的对象的监控状态和在自动机中不会到达终止状态的部分匹配；其次选择绑定的变量，对有限状态模型中状态的标记约束中的析取进行划分，从而减少更新状态的开销，将该划分以树的形式表示，还引入索引机制，提供了对部分匹配的快速访问。

文献[15]采用一种基于二叉决策图的静态分析在编绎时排除插装点，提出一种 PQL 语言，使得开发者更容易利用上

下文敏感指向分析的结果，在指向分析结果的基础上将 PQL 查询自动转换成查询，使用户不需要直接操作程序表示或上下文敏感分析结果。该分析计算非递归程序中每个调用路径的指向关系。对于递归程序，通过将每个强连通分量作为一个单独的结点对其调用路径进行简化。将指向信息存于一个演绎数据库中。这些数据是用二叉决策图表示的，使用逻辑编程语言写的查询可以对其进行有效的访问。

虽然使用上述的静态分析方法可以减少所需监控器的数量，然而这种技术是基于所监控的目标系统的，有如下两个缺点：首先它比较复杂，且速度比较慢；其次，它必须对每个目标程序进行分析，这使得该方法不能模块化。最后这些静态分析技术都是针对特定的规约形式，而监控系统通常需要能够支持多种逻辑形式来描述被验证的属性。

5.2 enable 集优化

针对静态程序分析方法的缺点，文献[23]提出基于 enable 集的参数化监控方法，通过查看已发生的事件，判断一个不完整的轨迹切片是否有可能达到一个期望的类别。如果不能到达期望的类别，则忽略该事件，忽略那些不可能到达目标类别的参数实例，从而减少创建的监控器实例，大大减少监控开销。

定义 1(轨迹 enable 集) 给定轨迹 $\tau\in\varepsilon^*$，且 $e,e'\in\tau$，用 $e'\vdash\!\!\!\rightarrow e$ 表示在轨迹 τ 中，事件 e' 早于事件 e 出现，则事件 e 的轨迹 enable 集表示为 $enable_\tau(e)=\{e'\mid e'\vdash\!\!\!\rightarrow e\}$。

定义 2(性质参数 enable 集) 给定性质 $P:\varepsilon^*\rightarrow C$，一个目标类别集 $G\subseteq C$ 和一个参数集 X，则事件 e 的性质参数 enable 集定义为：$enable_G^X(e)=\{\bigcup^X_{enable_\tau(e)}\mid P(\tau)\in G\}$。

事件 e 性质参数 enable 集是一组参数集，这些参数集中的每一个元素表示事件 e 发生前所出现的参数。如对于图 2 中的例子，若 $G=\{match\}$，则 $enable_G^x(create)=\{\emptyset\}$，$enable_G^x(next)=\{\{v,e\}\}$，$enable_G^x(update)=\{\{v,e\}\}$。

该方法的主要思想是：基于待监控的性质，可将轨迹划分为不同的类别(违背的轨迹、确认的轨迹、未知的轨迹等)，计算每个事件的参数 enable 集，在监控过程中，当观察到某一事件 e 时，比较新的监控状态的参数实例与事件 e 的 enable 集，如果该参数实例属于参数 enable 集，则创建一个新的监控器状态，否则，忽略该事件，避免不必要地创建一些永远不会触发的监控器。

通过计算事件的 enable 集来避免处理冗余的参数绑定，enable 集只依赖于性质，而不依赖于用于指定性质的规约形式。该方法已在 JAVAMOP[24] 中实现。

5.3 索引机制

由 4.2 节可知，在分散式参数化监控方法中，给每个参数实例维持一个监控器，就会产生许多监控器实例，对于每个接收的事件，主要是要快速找到所有与给定的一组参数实例相关的监控器，以更新它们的状态。因此，有效的监控器查找对于减少运行时开销是至关重要的。

现有的运行时监控系统大多采用集中式索引来实现有效的监控器查找，集中式索引是根据事件定义构造多个索引树来避免低效的索引遍历。对于规约中每个不同的事件参数集，创建一个索引树，将参数集直接映射到对应的监控器列表。集中索引的方法在运行时开销上是可接受的，然而减少运行时开销一直是运行时监控中所要考虑的问题，因此文献[25]中提出一个基于分散索引的优化方法。

在分散索引中，将索引树放于对象的状态中，以减少查找开销，对于每个不同的参数子集，这些参数作为某些事件的参数出现，系统自动选择其中的一个参数作为主参数，使用其它参数和哈希映射构建索引树，生成的映射将被声明为主参数的一个字段，用于存储运行时相关的监控器实例，这样就能直接通过参数的该字段获取相应的监控器实例。

集中式索引和分散式索引的主要区别是：使用分散式索引时，与参数相关的监控器列表可以直接由参数重新获取，从而避免不必要的查找操作，减少运行时开销和内存使用。而集中式索引需要从哈希映射中查找该列表。分散索引技术是逻辑独立的，但由于分散索引需要更改主参数的原始结构，因此分散索引相对于集中式索引涉及到更多的插装。

集中式索引和分散索引都采用了索引树结构，尽管索引树是监控中最常用且有效的数据结构，但当存在大量的事件时，索引树经过多级映射寻找相关监控器，可能会引起显著的开销。该开销来源于散列过程，由于索引树是由基于哈希映射的数据结构组成的，因此需在树的每一层进行散列操作。这样，通过索引树获取监控器的开销就远大于常规访问一个局部变量或对象域的开销。

针对上述问题，文献[26]引入索引缓存来存储每个索引树上一次获取的监控器，实验研究表明，通常将缓存块设为 1。对于接收的事件，如果我们缓存了需要更新的监控器实例，则只需要在最开始执行一次散列操作，后续通过该缓存直接获取监控器实例，这样大大减少了运行时监控开销。

5.4 监控器垃圾回收

对于分散式的参数化运行时监控，参数在运行时是动态绑定的，因而会产生大量的监控器实例，对于复杂的性质和系统来说，如何有效地回收监控器实例就成为必要了。

文献[13,28]提出一种垃圾回收监控器状态中变量绑定的方法，该方法使用 JAVA 中的弱引用对象，能保证在变量绑定失效时，删除相应的部分匹配。该方法将变量分成 3 类：collectableWeakRef、weakRefs 和 StrongRefs。其中从当前状态到终止状态的每条路径上都被绑定的变量属于 collectableWeakRef 集合，而没有在 tracemaatch 主体中使用且不属于 collectableWeakRef 集合的变量被划分到 weakRefs 集合中，其它的变量都属于 StrongRefs 集合。collectableWeakRef 中的变量足以让变量保持对绑定值的弱引用，且当该变量失效时，则删除整个部分匹配。WeakRef 中的变量也需要保持对变量值的弱引用。对于那些在到达终止状态的路径上不需要重新被绑定且用于 tracematch 主体中的变量，需保持对变量的强引用。这样确保当不再使用某一变量时，快速删除包含该变量绑定的部分匹配，从而避免了空间泄露，但该方法只针对正则表达式所表示的规约形式。

JAVAMOP[24] 提出一种规约形式独立的监控器垃圾回收方法。它只有在所有与某一监控器实例绑定的变量被垃圾回收后，才能收集相关的监控器实例，这样会导致不必要的监控器存在于整个程序过程中，且会产生内存泄露。

文献[27]在上述方法的基础上进行了一定的改进，提出一种懒散的监控器垃圾回收方法以避免产生不必要的开销。基于事件的 coenable 集，可消除不必要的监控器实例。

定义 3(轨迹 coenable 集) 给定轨迹 $\tau\in\varepsilon^*$，且 $e,e'\in\tau$，

用 $e \vdash\!\!\rightarrow e'$ 表示在轨迹 τ 中，事件 e' 出现于事件 e 之后，则事件 e 的轨迹 coenable 集表示为 $coenable_\tau(e)=\{e' \mid e \vdash\!\!\rightarrow e'\}$。

定义 4(性质 coenable 集) 给定性质 $P:\varepsilon^* \rightarrow C$，一个目标类别集，则性质 coenable 集可表示为 $coenable_{P,G}(e)=\{coenable_\tau(e) \mid \tau \in \varepsilon^* \text{ s. t. } P(\tau)\in G, e\in\tau, coenable_\tau(e)\neq\emptyset\}$。

定义 5(性质参数 enable 集) 给定性质 $P:\varepsilon^* \rightarrow C$，一个目标类别集 $G\subseteq C$，一个参数集 X 和事件定义 $D:\varepsilon\rightarrow P(X)$，则事件 e 的性质参数 enable 集定义为：$coenable^X_{P,G}(e)=\{D(E) \mid E\in coenable_{P,G}(e)\}$。

对于图 2 中的实例，若 $G=\{match\}$，计算每个事件的 coenable 集，$coenable^X_{P,G}(create)=\{\{c,i\}\}$，$coenable^X_{P,G}(next)=\{\{c,i\}\}$，$coenable^X_{P,G}(update)=\{\{i\},\{c,i\}\}$。

其主要思想是：将轨迹划分为不同的类别(违背的轨迹、确认的轨迹、未知的轨迹等)，计算每个事件的轨迹 coenable 集，与事件 e 相关的轨迹 coenable 集是一组事件集，但这些事件集中的每一个元素表示在该轨迹上发生于事件 e 之后的事件。然后再通过轨迹 coenable 集计算性质 coenable 集。在监控过程中，比较未来发生的事件集与当前事件 e 的 coenable 集，当事件 e 的 coenable 集的事件在未来不可能发生，则删除 e 的 coenable 集中所有不能使轨迹属于期望类别的事件集，同时也删除相应的监控器实例。coenable 集在运行时用来确定监控器实例什么时候再也不能满足期望的性质，从而可以被垃圾回收。然后采用一个懒散的垃圾回收方案，当访问一个包含被垃圾回收参数对象的索引树时，就告知它包含的所有相关的监控器。然后监控器使用 coenable 集确定哪些监控器实例可被垃圾回收，然后当数据结构中需要更多的空间或更新监控器时，就从访问的数据结构中去除监控器。如果一个数据结构本身被垃圾回收了，则其包含的监控器不需要单独进行垃圾回收。

通过上述方法回收不必要的监控器不仅减少了内存使用，还减少了更新监控器实例的时间，因为许多要更新的监控器不再是必要的。

结束语 参数化运行时监控是针对程序中的动态性质，监控和分析系统执行，检查其是否满足指定的参数化性质的技术。本文对现有的参数化性质规约、参数化运行时监控方法以及减少运行时开销的方法进行了介绍。从本文的介绍中可以看出，参数化运行时监控有丰富的理论价值和应用前景，是一个有意义的研究热点，值得深入研究。现有的研究取得了较好的进展，但还存在一些不足，该领域还存在如下一些值得进一步研究的问题：

1. 研究典型安全关键软件中参数化性质模板

以典型安全关键软件在运行时需要监控的性质为背景，研究这些典型安全关键软件中关键性质的参数化表征及相应动态性质的特点，结合并发、分布、实时、Web 服务及开发环境等关键软件的典型特性以及状态、时序、资源等各类监控需求，研究各类典型的监控性质共性，以定义参数化监控性质模板。

2. 研究形式独立的静态分析技术

本文所介绍的以及目前运行时监控系统所使用的程序静态分析技术都是针对特定的规约形式，但实际应用中，监控系统通常需要能够支持多种逻辑形式来描述被验证的属性。因此，可以研究形式独立的静态分析技术。

3. 研究面向参数化性质软件主动监控和调控技术

现有的参数化运行时监控的研究主要专注于高效的运行时监控算法以及如何减少运行时开销这两方面，而在实际应用中，更多的是要保证系统不会出现失效，因此需要研究参数化性质软件主动监控和调控技术，以及如何将该技术应用于软件设计和开发过程中。

参 考 文 献

[1] Clarke E M, Wing J M. Formal methods: State of the art and future directions[J]. ACM Computing Surveys, 1996, 28(4): 626-643

[2] Clarke E M, Grumberg O, Peled D A. Model Checking [M]. Massachusetts: The MIT Press, 1999

[3] Gabbay D M, Hogger C J, Robinson J A, et al. Handbook of Logic in artificial Intelligence and Logic Programming[C]// Volume2, Deduction Methodologies. London: Oxford University Press, 1994

[4] Blum M, Kannan S. Designing programs that check their work [J]. Journal of the ACM, 1995, 42(1): 269-291

[5] Bauer A, Leucker M, Schallhart C. Runtime Verification for LTL and TLTL[J]. ACM Transactions on Software Engineering and Methodology, 2011, 20(4): 1-64

[6] Havelund K, Rosu G. Synthesizing Monitors for Safety Properties[C]// Proceedings of the 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems. London, UK, 2002: 342-356

[7] Bodden E, Hendren L, Lam P, et al. Collaborative runtime verification with tracematches[C]// Proceedings of the 7th international conference on Runtime Verification. Berlin, Germany, 2007: 22-37

[8] Bodden E. A lightweight LTL runtime verification tool for Java [C]// Proceedings of the 9th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. New York, USA, 2004: 306-307

[9] Havelund K, Rosu G. Monitoring Java Program with Java Path Explorer[J]. Electronic Notes in Theoretical Computer Science, 2001, 55(2): 200-217

[10] Kim M, Kannan S, Lee L, et al. Java-MaC: a run-time assurance approach for Java programs[J]. Formal Methods in System Design, 2004, 24(2): 129-155

[11] 赵常智，董威，隋平，等. 面向参数化 LTL 的预测监控器构造技术[J]. 软件学报，2010，21(2)：318-333

[12] Bodden E. J-LO-A tool for runtime-checking temporal assertions [D]. Germany: RWTH Aachen University, 2005

[13] Avgustinov P, Tibblw J, de Moor O. Making Trace Monitors Feasible[C]// Proceedings of the 22nd Annual ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications. New York, USA, 2007: 589-608

[14] Alan C, Avgustinov P, Christensen A S, et al. A dding trace matching with free variables to AspectJ[C]// Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications. New York, USA, 2005: 345-364

(下转第 174 页)

Exact signal reconstruction from highly incomplete frequency information[J]. IEEE Transactions on Information Theory, 2006, 52(2): 489-509

[2] Donoho D. Compressed Sensing[J]. IEEE Transactions on Information Theory, 2006, 52(4): 1289-1306

[3] 石光明,刘丹华,高大化,等. 压缩感知理论及其研究进展[J]. 电子学报, 2009, 37(5): 1070-1081

[4] Candes E, Tao T. Decoding by Linear Programming[J]. IEEE Transactions on Information Theory, 2005, 51(12): 4203-4215

[5] Tropp J, Gilbert A. Signal Recovery from Random Measurements via Orthogonal Matching Pursuit[J]. IEEE Transactions on Information Theory, 2007, 53(12): 4655-4666

[6] Needell D, Vershynin R. Uniform Uncertainty Principle and Signal Recovery via Regularized Orthogonal Matching Pursuit[J]. Foundations of Computational Mathematics, 2009, 9(3): 317-334

[7] Donoho D, Tsaig Y, Drori I, et al. Sparse Solution of Underdetermined Systems of Linear Equations by Stagewise Orthogonal Matching Pursuit[J]. IEEE Transactions on Information Theory, 2012, 58(2): 1094-1121

[8] Dai W, Milenkovic O. Subspace Pursuit for Compressive Sensing Signal Reconstruction[J]. IEEE Transactions on Information Theory, 2009, 55(5): 2230-2249

[9] Needell D, Tropp J. CoSaMP: Iterative Signal Recovery from Incomplete and Inaccurate Samples[J]. Applied and Computational Harmonic Analysis, 2009, 26(3): 301-321

[10] Do T, Lu G, Nguyen N, et al. Sparsity Adaptive Matching Pursuit Algorithm for Practical Compressed Sensing[C]// Pacific Grove. Conference Record of the Asilomar Conference on Signals, Systems and Computers. California: IEEE, 2008: 581-587

[11] Maleki A, Donoho D. Optimally Tuned Iterative Reconstruction Algorithms for Compressed Sensing[J]. Selected Topics in Signal Processing, IEEE Journal of, 2010, 4(2): 330-341

[12] 姚远,梁志毅. 基于压缩感知信号重建的自适应空间正交匹配追踪算法[J]. 计算机科学, 2012, 39(10): 50-53

[13] Karahanoglu N, Erdogan H. Compressed sensing signal recovery via forward-backward pursuit[J]. Digital Signal Processing, 2013, 23(5): 1539-1548

(上接第 151 页)

[15] Martin M, Livshits B, Lam M S. Finding Application Errors and Security Flaws Using PQL: a Program Query Language[C]// Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. New York, USA, 2005: 365-383

[16] Goldsmith S, O'Callahan R, Aiken A. Relational Queries Over Program Traces[C]// Proceedings of the 20th Annual ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, And Applications. New York, USA, 2005: 385-402

[17] Barringer H, Goldberg A, Havelund K, et al. Rule-Based Runtime Verification[C]// Proceedings of the 5th International Conference on Verification, Model Checking, and Interpretation (VMCAI 2004). Venice, Italy, 2004: 44-57

[18] Barringer H, Rydeheard D, Havelund K. Rule systems for runtime monitoring: from EAGLE to RULER[C]// Proceedings of the 7th International Workshop on Runtime Verification. Vancover, Canada, 2007: 111-125

[19] Dwyer M B, Purandare R. Residual dynamic typestate analysis exploiting static analysis: results to reformulate and reduce the cost of dynamic analysis[C]// Proceedings of the 22nd IEEE/ACM International Conference on Automated Software Engineering. New York, USA, 2007: 124-133

[20] Bodden E, Hendren L, Lhotak O. A staged static program analysis to improve the performance of runtime monitoring[C]// Proceedings of the 21st European conference on Object-Oriented Programming. Berlin, Germany, 2007: 525-549

[21] Bodden E, Feng Chen, Rosu G. Dependent Advice: A General Approach to Optimizing History-based Aspects[C]// Proceedings of the 8th International Conference on Aspect-Oriented Software Development. Virginia, USA, 2009: 3-14

[22] Bodden E, Lam P, Hendren L. Clara: a Framework for Partially Evaluating Finite-state Runtime Monitors Ahead of Time[C]// Proceedings of the First International Conference on Runtime Verification. Julians, Malta, 2010: 183-197

[23] Chen Feng, Jin Dong-yun, Meredith P O, et al. Efficient Formalism-Independent Monitoring of Parametric Properties[C]// Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering. Washington, USA, 2009: 383-394

[24] Jin Dong-yun, Meredith P O, Lee C, et al. JavaMOP: efficient parametric runtime monitoring framework[C]// Proceedings of the 2012 International Conference on Software Engineering. Piscataway, USA, 2012: 1427-1430

[25] Chen Feng, Rosu G. MOP: An Efficient and Generic Runtime Verification Framework[C]// Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. New York, USA, 2007: 569-588

[26] Jin Dong-yun. Making Runtime Monitoring of Parametric Properties Practical[D]. USA: University of Illinois at Urbana-Champaign, 2012

[27] Jin Dong-yun, Meredith P O, Griffith D, et al. Garbage Collection for Monitoring Parametric Properties[C]// Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA, 2011: 415-424

[28] Avgustinov P, Tibble J, Bodden E, et al. Efficient Trace Monitoring[C]// Proceedings of the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications. New York, USA, 2006: 685-686

[29] Meredith P O, Jin Dong-yun, Chen Feng, et al. Efficient Monitoring of Parametric Context-Free Patterns[J]. Automated Software Engineering, 2010, 17(2): 149-180

[30] Rosu G, Chen Feng. Semantics and Algorithms for Parametric Monitoring[J]. Logical Methods in Computer Science, 2012, 8(1): 1-47

[31] Chen Feng, Rosu G. Parametric Trace Slicing and Monitoring[C]// Proceeding of the 15th International Conference on Tools and Algorithms for the Construction and Analysis of the Systems. Berlin, Germany, 2009: 246-261