基于数据库事务的不变式推导

曾虹驰[1] 彭 鑫[1,2] 赵文耘[1,2]
(复旦大学软件学院 上海 201203)[1] (复旦大学上海市数据科学重点实验室 上海 201203)[2]

摘 要 作为数据处理和并发控制的基本单位,数据库事务被广泛应用于软件系统的业务逻辑中。通过收集运行时数据库事务中的数据,推导这些数据之间满足的不变式,建立相应的数据契约关系,是软件维护过程中对系统的内部状态进行监控的重要方法之一。目前,在不变式推导领域,主要的方法和工具都是基于代码进行分析的,缺少与基于数据分析相关的研究和成果。为了解决这一问题,首先提出了基于数据的推导代数等式形式的不变式的算法,然后设计并实现了基于数据库事务的不变式推导的原型工具,最后通过相关实验分析和验证了原型工具的有效性。实验结果表明,原型工具有良好的推导准确率和运行性能,能够弥补现有工具和方法在基于数据的分析领域的不足。

关键词 数据库事务,不变式推导,代数等式
中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2017.11.014

Deriving Invariants from Database Transactions

ZENG Hong-chi[1] PENG Xin[1,2] ZHAO Wen-yun[1,2]
(Software School, Fudan University, Shanghai 201203, China)[1]
(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)[2]

Abstract As the basic unit of data processing and concurrency control, database transaction is widely used in business logic of software applications. It is a significant method in software maintenance for monitoring internal state to derive invariants and establish corresponding data contracts based on data collected from database transactions at runtime. Currently, in the field of invariant derivation, most researches and tools are just based on deriving invariants from source code, while there is lack of researches on deriving invariants from data. To solve this problem, this paper firstly introduced an algorithm for deriving invariants in the form of algebraic equations. Moreover, a prototyping tool was designed and implemented to collect data from database transactions at runtime and derive invariants among these data. With experiments, the prototyping tool is proved to be effective both on accuracy and performance, which fills the gap of invariants derivation based on data.

Keywords Database transaction, Invariant derivation, Algebraic equation

1 引言

随着计算机在金融、工业、通信、医疗等行业的应用日趋广泛,计算机软件的系统设计和逻辑流程变得愈加复杂,由于软件的复杂性而导致的软件缺陷和漏洞问题日益突出。一方面,软件漏洞使系统存在被恶意攻击的风险,导致系统数据被非法篡改和窃取,使用户权益遭受侵害;另一方面,软件缺陷的存在将引起系统在关键路径的执行和计算上出现错误,导致严重的系统故障和安全事故,对开发者造成重大损失。因此,对软件系统进行监控和分析,挖掘其中可能存在的缺陷和漏洞,成为了软件维护领域亟待研究和解决的重要问题之一。

互联网大数据时代的到来使大数据在学术界和工业界得到广泛应用,同时也使基于数据的缺陷和漏洞分析进入人们的视野当中。大数据自身存在的规模性、高速性、多样性和价值稀疏性的特点,使其有可能存在数据质量问题,如数据不一致、不精确、不完整、过时等[1]。因此,对软件运行时产生的数据进行收集和监控,基于这些数据分析其中可能存在的质量问题,对于挖掘软件系统中潜在的缺陷与漏洞以及构建高可信的稳定系统起着重要的作用[2]。

目前对软件缺陷和漏洞的分析主要包括架构安全分析、源代码漏洞分析、二进制漏洞分析等[3],但这些分析方法都是基于软件架构或源代码进行的,在基于数据的分析上缺乏适用性。为了解决这一问题,本文试图通过收集系统运行时的数据,推导出数据之间隐含的不变式,构建相应的数据契约关系,以验证数据的一致性和可靠性。该方法的意义主要表现在以下两个方面:

到稿日期:2016-10-03 返修日期:2016-12-18 本文受国家自然科学基金(61370079)资助。
曾虹驰(1994-),男,硕士生,主要研究方向为软件工程、自适应软件体系、移动计算与云计算;彭 鑫(1979-),男,教授,CCF高级会员,主要研究方向为需求工程、自适应软件、软件产品线、软件维护、逆向工程,E-mail:pengxin@fudan.edu.cn;赵文耘(1964-),男,教授,CCF高级会员,主要研究方向为软件复用、软件产品线、软件工程。

(1)软件缺陷挖掘:在设计任何软件时,其业务逻辑处理的数据之间必然存在某种数据契约关系。如果对运行时数据库事务的数据契约关系进行推导后得到的不变式关系不符合设计预期,则说明软件系统可能存在设计或编码缺陷。此时开发者可以根据相关数据日志进行追溯,挖掘代码中存在的缺陷和问题。

(2)数据一致监控:软件在运行过程中,由于受到恶意攻击或运行环境出现变化,会产生少量的异常数据,这些数据表现为不满足事务中原本存在的契约关系。因此,基于历史正常数据推导出的不变式可以用于运行时的数据异常监控和分析,以便对可疑数据做出及时的预警和应对。

本文的主要贡献:提出了对代数等式形式的不变式的推导算法;实现了基于数据库事务进行不变式推导的原型工具;通过对原型工具进行实验和结果分析,探讨了在基于数据进行不变式分析领域未来可能的进一步研究方向。

本文第 2 节简述研究的相关工作;第 3 节提出不变式推导的算法;第 4 节论述不变式推导原型工具的设计和实现;第 5 节进行实验并对结果进行分析和探讨;最后总结全文并展望未来可能的研究方向。

2 相关工作

不变式的概念最早出现于契约式设计(Design by contract)[4]领域,作为形式化方法中证明程序的正确性的重要手段,它在程序和数据的分析中起着关键作用。

依据不变式约束的对象所处的位置,可以将不变式分为前置条件、后置条件、循环不变式、类不变式等类型。由于不变式可以对程序的各个位置进行约束,使其可以表现程序执行的各种特征,包括程序的运行逻辑、数据结构等,因此在早期的软件开发和维护过程中,不变式主要以语法扩展的方式应用于源代码的编写中。典型的支持不变式的语言或开发框架包括 Eiffel[5],JML(Java Modeling Language)[6]等。这些编程语言或开发框架支持开发者在变量、函数、类等位置显式地插入不变式作为断言,从而在程序运行时监控其是否始终满足给定的不变式。

随着软件结构和逻辑日益复杂,开发者手动编写不变式的方式表现出了效率低、准确性差、可读性弱等缺点[7],由此产生了许多自动化的不变式推导工具,例如 Daikon[8],Celeriac[9]和 Diduce[10]等。这些工具通过在源代码中的特定位置插入跟踪点来分析和记录运行时所关注的变量的值的变化,从而推导出变量之间满足的不变式关系。其中,Daikon 和 Celeriac 支持推导单一变量自身的取值范围、最大(小)值、枚举值集合等,以及推导多个变量(不超过 3 个)之间满足的代数等式和代数不等式关系[11]。Diduce 仅支持单一变量在函数执行前后自身满足的不变式关系。

对于多数软件系统来说,业务逻辑中的数据库事务涉及的变量通常比较多,并且变量间的不变式关系不限于简单的线性关系。而目前主要的动态不变式推导工具并不支持推导超过 3 个以上的变量之间的线性关系和复杂的代数等式关系,故本文试图对这一问题进行研究和探讨,以弥补相关技术上的空白。

3 不变式推导方法

对于不同数学形式的不变式,存在着不同的推导算法。在软件系统中,由于代数等式形式的不变式具有较高的出现概率,因此本文选择对这一形式的不变式的推导进行研究和探讨。本节首先给出对于数据来源的软件系统的基本假设,然后提出基于线性代数的不变式推导算法。

3.1 基本假设

本文在进行不变式推导时,对数据来源的软件系统作出如下假设。

假设 1(存在数据契约) 只有软件系统的业务逻辑中存在某种确定的数据契约关系时,才有可能推导出其对应的不变式;否则,如果业务逻辑本身具有随机性和不确定性,进行不变式推导将会变得毫无意义。

假设 2(遵循强事务体系) 强事务体系要求系统执行的每个事务独立完整地实现某个业务逻辑流程,使得数据契约以不变式的形式存在于各个事务内部。与之相对的是最终一致性事务体系,在这种体系下,多个事务共同完成特定的业务逻辑,导致数据契约无法独立于某一个特定的事务存在,从而极大地增加了不变式推导的复杂性。

假设 3(通常情况下正常运行) 由于不变式的推导需要基于系统运行时产生的数据进行采样和分析,因此系统在大多数情况下正常工作并产生正确的内部数据,这是原型工具能够准确推导不变式的必要条件之一。如果软件系统本身以极大的概率产生异常数据,则将对数据分析造成干扰,导致不变式的最终推导结果出现偏差。

3.2 推导算法

本节基于线性代数的相关理论[12],通过构造和求解线性方程组[13],设计了针对代数等式形式的不变式的推导算法。其中,3.2.1 节形式化地定义了不变式的求解问题,3.2.2 节提出了相应的推导算法来求解代数等式形式的不变式。

3.2.1 算法问题定义

对于数值类型的数据,其代数等式形式的不变式推导问题的形式化定义如下。

定义 1(不变式推导问题) 对于 n 个数值类型的变量 $x_1,x_2,\cdots,x_n$,将其按照一定顺序排列,构成变量向量 $X=(x_1,x_2,\cdots,x_n)$。由 X 中每一个变量 $x_i(1\leqslant i\leqslant n)$ 的值 v_i 构成的向量 $V=(v_1,v_2,\cdots,v_n)$ 称为 X 的一个值向量。给定一个变量向量 X 和它的若干值向量 $V_1,V_2,\cdots,V_n$,如果这些值向量 $V_i(1\leqslant i\leqslant n)$ 中存在不小于阈值 $\alpha(0\leqslant\alpha\leqslant 1)$ 的比例使得“变量 $x_j(1\leqslant j\leqslant n)\in V_i$ 之间满足某个代数等式”成立,即存在非平凡解 $a_1,a_2,\cdots,a_m$(a_i 不全为 0)使得式(1)成立,则称 X 中的变量之间存在一个代数等式形式的不变式。

$$\sum_{i=1}^{m}a_it_i=0,t_i=\prod_{k=1}^{n}x_k^{e_{i,k}}(e_{i,k}\in N,x_k\in X) \tag{1}$$

基于该定义,给定变量向量 X、值向量集合 $\{V_n\}$ 和比例阈值 α,不变式推导算法的目标即为求出使式(1)成立的非平凡解 $a_1,a_2,\cdots,a_m$。

3.2.2 算法执行流程

3.2.1 节中定义的不变式推导问题可以通过线性代数对方程组进行消元和化简,从而得到方程的解。该算法的主要流程如下。

(1)构造方程组。从值向量集合 $\{V_n\}$ 中任意选取 m 个值向量 $V_1,V_2,\cdots,V_m$,对于每一个 $V_i(1\leqslant i\leqslant m)$,将其中的变量值 $v_{i,1},v_{i,2},\cdots,v_{i,m}$ 代入式(1)中,计算系数 $t_{i,j}=\prod_{k=1}^{m}x_k^{e_{j,k}}=\prod_{k=1}^{m}$

$v_{i,k}^{e_{j,k}}(e_{j,k}\in N)$，得到关于 $a_1,a_2,\cdots,a_m$ 的含有 m 个方程的线性方程组。该方程组的系数矩阵为 $m\times m$ 的方阵，如式(2)所示。

$$\begin{pmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,m} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ t_{m,1} & t_{m,2} & \cdots & t_{m,m} \end{pmatrix} \tag{2}$$

(2)消去常系数项。为了简化方程组，对于具有常系数 $t_{i,j}=\prod_{k=1}^{m}x_0^k=1(1\leqslant i\leqslant m,1\leqslant j\leqslant m)$ 的项 $t_{i,j}a_{i,j}$，将其从每个方程中移除。为了保持系数矩阵为方阵，从方程组任选一个方程，使用加减消元的方式，用其他所有方程减去此方程，然后将该方程从方程组移除。设被移除的常系数项为 $a_{i,j}$，被移除的方程为系数矩阵中第 k 行对应的方程，则消去常系数项后方程组的系数矩阵为 $(m-1)\times(m-1)$ 的方阵，如式(3)所示。

$$\begin{pmatrix} t_{1,1}-t_{k,1} & \cdots & t_{1,j-1}-t_{k,j-1} & t_{1,j+1}-t_{k,j+1} & \cdots & t_{1,m}-t_{k,m} \\ t_{2,1}-t_{k,1} & \cdots & t_{2,j-1}-t_{k,j-1} & t_{2,j+1}-t_{k,j+1} & \cdots & t_{2,m}-t_{k,m} \\ \vdots & & \vdots & \vdots & & \vdots \\ t_{k-1,1}-t_{k,1} & \cdots & t_{k-1,j-1}-t_{k,j-1} & t_{k-1,j+1}-t_{k,j+1} & \cdots & t_{k-1,m}-t_{k,m} \\ t_{k+1,1}-t_{k,1} & \cdots & t_{k+1,j-1}-t_{k,j-1} & t_{k+1,j+1}-t_{k,j+1} & \cdots & t_{k+1,m}-t_{k,m} \\ \vdots & & \vdots & \vdots & & \vdots \\ t_{m,1}-t_{k,1} & \cdots & t_{m,j-1}-t_{k,j-1} & t_{m,j+1}-t_{k,j+1} & \cdots & t_{m,m}-t_{k,m} \end{pmatrix} \tag{3}$$

(3)判断非平凡解的存在性。判断式(3)中的矩阵是否为奇异矩阵。如果矩阵为奇异矩阵，则原方程组存在一组非平凡解，继续执行算法的后续步骤；否则，矩阵为非奇异矩阵，原方程组不存在非平凡解，算法中止。

(4)消去无关主元。对于系数矩阵为 $m_1\times m_1$ 的矩阵 M_1 的方程组，设其中的主元构成的集合为 $A_1=\{a_{k_1},a_{k_2},\cdots,a_{k_m}\}(1\leqslant k_i\leqslant m)$。任意选取一个主元 $a_{k_i}\in A_1$，从每个方程中移除关于 a_{k_i} 的项，然后从方程组中任意移除一个方程，可以得到一个新的方程组，其系数矩阵为 $(m_1-1)\times(m_1-1)$ 的矩阵 M_2。如果 M_2 为奇异矩阵，则说明消去的是无关主元，此时继续对系数矩阵 M_2 构成的方程组消去无关主元，即尝试寻找其他可以消去的无关主元 a'_{k_i}；如果 M_2 为非奇异矩阵，则说明消去的不是无关主元，此时保留主元 a_{k_i}，尝试从 M_1 中消去其他无关主元 a'_{k_i}。当且仅当方程组的任意一个主元 a_{k_i} 都无法被消去时，中止消元操作，执行算法的后续步骤。

(5)求方程组的特解。对于经过消去无关主元后得到的方程组，从中任意选取一个主元 a^* 作为自由变量，令 $a^*=1$，然后从方程组中任意移除一个方程，得到一个具有唯一特解的线性方程组。使用高斯消元法求解该线性方程组，记它的解为行向量 A^*。

(6)求常系数项的解。从原方程组中任选一个方程，对于其中每一个主元 $a_i(1\leqslant i\leqslant m)$，若 $a_i\in A^*$，令其取对应的值，否则 $a_i=0$。此时，该方程成为一个关于步骤(2)中被消去的常系数项 $a_{i,j}$ 的一元一次方程。求解该方程得到 $a_{i,j}$ 的值，结合 A^* 即可得到原方程组的一个解向量 $A=(a_1,a_2,\cdots,a_n)$。

(7)阈值验证。由解向量 A 可以构造一个形如式(1)的代数等式，将值向量集合 $\{V_n\}$ 中的每一组向量 V_i 代入并计算 $t_{i,j}$，验证等式是否成立。如果存在超过 $|\{V_n\}|\times\alpha$ 个值向量满足该代数等式，则可以确认由 A 构成的代数等式为变量向量 X 中存在的不变式。

4　不变式推导工具的设计与实现

本文利用上文提出的不变式推导算法实现了一个基于数据库事务的不变式推导原型工具(简称原型工具)。原型工具通过对软件运行过程中的数据库事务进行收集和分析，推导出数据之间存在的代数等式形式的不变式关系。本节首先给出在设计原型工具时所提出的基本定义，然后介绍原型工具各个模块的具体实现和工作流程。

4.1　基本定义

定义 2(字段名)　字段名 f 为 SQL 语句的操作列表中某个字段的名称。为了区分不同数据表，采用“表名.字段名”的形式进行记录。对于图 1 所示的数据库事务代码，第一条 SQL 语句 s_1 的操作字段列表中共含有 3 个字段，其字段名分别为 $f_1=$“*demo.B*”，$f_2=$“*demo.C*”，$f_3=$“*demo.D*”。

定义 3(字段名向量)　字段名向量 F 为 SQL 语句中所有的字段名 f_i 构成的向量，即 $F=(f_1,f_2,\cdots,f_n)$。对于图 1 所示的数据库事务代码，s_1 的字段名向量 $F_1=(f_1,f_2,f_3)=$(“*demo.B*”,“*demo.C*”,“*demo.D*”)。

定义 4(字段值)　字段值 x 为 SQL 语句的操作列表中某个字段对应的取值。对于图 1所示的数据库事务代码，s_1 中的 3 个操作字段的字段值依次为 $x_1=30,x_2=15,x_3=45$。

定义 5(字段值向量)　字段值向量 X 为 SQL 语句中所有的字段值 x_n 构成的向量，即 $X=(x_1,x_2,\cdots,x_n)$。对于图 1 所示的数据库事务代码，s_1 的字段值向量为 $X_1=(x_1,x_2,x_3)=(30,15,45)$。

定义 6(SQL 语句框架)　SQL 语句框架 p 描述了一条 SQL 语句的核心骨架，它保留了 SQL 语句的操作类型(SELECT，INSERT，UPDATE，DELETE)、操作字段列表和操作值占位符，同时移除其他与不变式推导无关的语法内容，如 WHERE 子句。对于图 1 所示的数据库事务代码，其中的两条 SQL 语句 s_1 和 s_2 对应的 SQL 语句框架 p_1 和 p_2 均为“insert into demo (B,C,D) values(?,?,?)”。

```
insert into demo(B,C,D) values (30,15,45);//s1
insert into demo(B,C,D) values (70,25,75);//s2
```

图 1　包含两条 SQL 语句的数据库事务

定义 7(事务模板)　事务模板 $T(N,\{p_n\},F,L)$ 由模板编号 N、SQL 语句框架集合 $\{p_n\}$、字段名向量 F 和字段值向

量组 L 构成。其中，N 为模板创建时为其分配的唯一编号，$\{p_n\}$为事务包含所有的 SQL 语句框架，F 为$\{p_n\}$中所有的字段名构成的向量，L 为事务多次执行过程中产生的不同的字段值向量 $X_1, X_2, \cdots, X_n$ 构成的向量组。它用于描述一个特定的业务逻辑对应的数据库事务的特征，即对于同一段数据库事务代码，当它被多次执行时，每一次收集到的数据信息都应当保存在同一个事务模板中。

定义 8(数值变量) 数值变量 v 为 SQL 语句中操作的数值型字段。对于一个数据库事务，如果在运行时某个字段的操作值始终为数值型，则将该字段视为数值变量。

定义 9(数值变量向量) 数值变量向量 V 为 SQL 语句中所有数值变量 v 构成的向量，即 $V=(v_1, v_2, \cdots, v_n)$。

定义 10(项) 对于一个数值变量向量 V，其中若干个数值变量 v_i 的非负整数幂相乘的积 $\prod_{i=1}^{|V|} v_i^{e_i}\ (e_i \in N, v_i \in V)$ 称为一个项 t。特别地，当 $e_1=e_2=\cdots=e_n=0$ 时，该项为常数项 $t=1$。将若干个项相加即可构成一个代数等式。

定义 11(幂向量) 对于一个项 $t=\prod_{i=1}^{|V|} v_i^{e_i}\ (e_i \in N, v_i \in V)$，由其中每个数值变量 v_i 的指数 e_i 构成的向量 $(e_1, e_2, \cdots, e_n)$ 称为项 t 的幂向量 E。

定义 12(度) 对于一个幂向量 $E=(e_1, e_2, \cdots, e_n)$，其中所有元素之和 $\sum_{i=1}^{n} e_i$ 称为该项的度 d。特别地，对于常数项 $t=1$，它的度 $d=0$。

定义 13(最大幂向量集) 给定度 d 和变量向量 V，可以构造出一个元素数目为 $C_{|V|+d}^{d}$ 的幂向量集合 $M(d, |V|)$，且其中每一个幂向量 $E_i\ (1 \leqslant i \leqslant |M|)$ 的度 d_i 满足 $d_i \leqslant d$。例如，对于 $d=2$ 和 $V=(v_1, v_2, v_3)$，可以构造出的最大幂向量集为 $M(d, |V|)=(0,0,0),(0,0,1),(0,0,2),(0,1,0),(0,1,1),(0,2,0),(1,0,0),(1,0,1),(1,1,0),(2,0,0)\}$。

4.2 原型工具的实现框架

原型工具通过对软件运行过程中的数据库事务语句进行收集和分析，推导出数据之间存在的不变式关系，并据此对数据异常进行监控和分析，确保系统在运行时的数据一致性和可靠性。如图 2 所示，原型工具由数据收集模块、数据提取模块和数据分析模块构成。其中数据收集模块通过跟踪数据库事务，收集事务的相关信息和其中包含的所有 SQL 语句；数据提取模块针对事务中的 SQL 语句结构和内容，提取其中操作的字段和对应的值，并构建相应的事务模板，用于后续分析；数据分析模块对事务中的数据进行分析，通过求解线性方程组，推导出数据之间可能存在的代数等式形式的不变式关系。

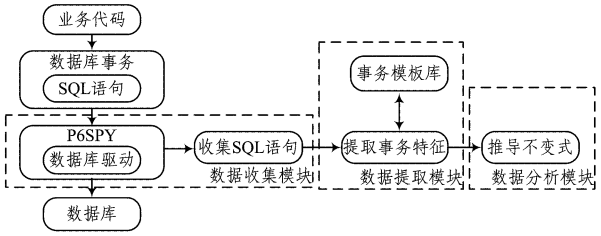

图 2 原型工具的工作原理示意图

4.2.1 数据收集模块

数据收集模块的目标是跟踪系统在运行时执行的数据库事务，在不影响事务正常执行的前提下，收集与事务执行相关的数据库连接信息、实际执行的 SQL 语句、事务执行结果等信息，并将这些信息传递给后续的模块用于数据提取。

本模块主要利用数据动态监测框架 P6Spy[14] 实现对系统运行时数据库事务的跟踪。P6Spy 是一个对数据库访问和操作进行动态监测的开源框架，它通过对 JDBC 驱动进行封装，在不需要对现有的程序进行代码修改和重新编译的前提下实现了对数据库相关操作的无缝跟踪和操控[15]。

为了提高对数据库事务中相关数据的跟踪和收集效率，本模块在实现过程中对 P6Spy 的实现代码进行了如下改动：

(1)以数据库事务为单位进行数据收集。P6Spy 默认以 SQL 语句为单位进行跟踪，而原型工具基于数据库事务进行不变式分析，其数据分析的最小单位是数据库事务。因此本模块对 P6Spy 的输出进行了批处理操作，将数据库事务包含的所有 SQL 语句进行集中处理，以提高数据收集的效率。

(2)过滤无关的操作语句。P6Spy 默认跟踪运行时执行的所有数据库操作，而原型工具主要关注的是与数据相关的操作。因此，本模块对 P6Spy 的跟踪对象进行了过滤，使其仅对数据库事务中的插入(INSERT)、更新(UPDATE)和删除(DELETE)语句进行跟踪，排除无关语句的干扰。

(3)重定向输出。P6Spy 默认仅支持将收集到的 SQL 语句输出到控制台或指定文件中，而本模块在实现过程中，将其重定向到原型工具的后续模块，以保证后续的不变式分析步骤能顺序进行。

基于以上改动，本模块对修改后的 P6Spy 项目代码进行了重新编译，然后封装成 JAR 包。在运行时，只需要使用该 JAR 包替换 JDBC 数据库驱动的 JAR 包，并修改数据连接配置文件中的相关参数，即可实现对运行时数据库事务的捕获。

例如，在运行图 3 所示的代码时，本模块通过 P6Spy 收集并重定向输出到后续模块的相关数据，如图 4 所示。

```
// insert into demo (B,C,D) values(30,15,45);
// insert into demo (B,C,D) values(70,25,75);
String sql = "insert into demo (B,C,D) values(?,?,?)";
PreparedStatement pstmt;
pstmt = conn.prepareStatement(sql);
pstmt.setString(1, String.valueOf(30));
pstmt.setString(2, String.valueOf(15));
pstmt.setString(3, String.valueOf(45));
pstmt.executeUpdate();
pstmt.setString(1, String.valueOf(70));
pstmt.setString(2, String.valueOf(25));
pstmt.setString(3, String.valueOf(75));
pstmt.executeUpdate();
conn.commit();
pstmt.close();
conn.close();
```

图 3 数据库事务代码

```
connection 1 | insert into demo (B,C,D) values(?,?,?) |
insert into demo (B,C,D) values('30','15','45');
connection 1 | insert into demo (B,C,D) values(?,?,?) |
insert into demo (B,C,D) values('70','25','75');
```

图 4　数据收集模块捕获的 SQL 语句信息

图 4 中，前两行为第一条 SQL 语句，后两行为第二条 SQL 语句。对于其中每一条 SQL 语句，从左至右依次为其对应的数据库的连接 ID、PreparedStatement 语句和完整的 SQL 语句。

4.2.2　数据提取模块

数据提取模块旨在对运行时数据库事务中的 SQL 语句进行数据和结构上的分析，生成相应的事务模板，记录事务中包含的字段和数据信息。

如图 5 所示，该模块的执行流程如下：

(1)对于数据收集模块中捕获的数据库事务信息，从中提取所有的 SQL 语句从而构成 SQL 语句集合$\{s_n\}$。对于$\{s_n\}$中的每一条语句 s_i，构造对应的 SQL 语句框架 p_i，从而得到 SQL 语句框架集合$\{p_n\}$。

(2)在事务模板库中，查询$\{p_n\}$是否已存在于某个事务模板 T 中，即该事务对应的事务模板 T 是否已被创建。如果不存在，则使用$\{p_n\}$构建新的事务模板 T，然后从$\{p_n\}$中提取所有字段名 f_i 构成字段名向量 F，并将其作为事务模板中 $T.F$ 的值。

(3)从$\{s_n\}$中提取所有的字段值 $x_1, x_2, \cdots, x_n$ 构建字段值向量 X，使得对于 F 中每一个字段名为 $f_i(1 \leqslant i \leqslant n)$的字段，在 SQL 语句中其对应的取值为 x_i。然后，将 X 添加到事务模板 T 的字段值向量组 $T.L$ 中。

(4)将事务模板 T 的模板编号 $T.N$ 传递给后续模块，用于进行不变式的推导。

```
algorithm EXTRACT-DATA
input: message M from data collection module
output: template index N
{sn},{pn}←fetch SQL statements from M
T←search for {pn} in template library
if (T=nil)
    let T be a new template
    T.N ← a unique template index
    T.pn← {pn}
    T.F ← {f1,f2,…,fn}, where fi occurs in {sn}
    let X be a new value vector
      for i=1 to F.length
      xi←value of field with name fi in {sn}
      insert xi into X
    insert X into T.L
return T.N
```

图 5　数据提取模块的逻辑流程伪代码

4.2.3　数据分析模块

数据分析模块旨在对数据库事务中的数据进行分析，推导出数据之间可能存在的不变式关系。对于数值类型的变量，本模块基于上文中的不变式推导算法，通过求解线性方程组来推导字段之间存在的代数等式形式的不变式关系。

本模块对不变式推导的逻辑流程如下：

(1)对于数据提取模块中传入的事务模板编号 N，从事务模板库中获得相应的事务模板 $T(N, \{p_n\}, F, L)$。

(2)对于$\{p_n\}$中的每一个字段名为 p_i 的字段，在每个字段值向量 $X_j \in L$ 中，检查该字段对应的字段值 $x_{i,j}$ 是否为数值型变量。如果所有字段值 $x_{i,j}$ 均为数值型，则将该字段作为一个数值变量添加到数值变量向量 V 中；否则，在进行不变式推导时，排除该字段。

(3)利用给定的度 d 和步骤(2)中得到的数值变量向量 V 计算最大幂向量集 $M(d, |V|)$。设 $|M|=m$，此时可以构造一个含有 m 项的代数等式，如式(4)所示。

$$t_1 a_1 + t_1 a_2 + t_1 a_3 + \cdots + t_m a_m = 0 \tag{4}$$

(4)使用 3.2.2 节中设计的算法求解式(4)中的方程组，即可推导出变量间存在的一个不变式。

(5)重复步骤(3)和步骤(4)若干次，取出现次数最多的不变式作为最终的推导结果。

5　实验与分析

通过构造存在不变式的数据库事务，本文对原型工具的不变式推导进行了相关实验，并将结果与 Daikon 进行对比，以检验和分析原型工具的实际推导效果和性能。本节首先对实验方法进行简要介绍，然后针对原型工具的不变式推导质量和运行性能进行实验，并对相关结果进行探讨和分析。

5.1　实验方法

实验首先随机生成若干组满足某个不变式的数据集，其中每一组数据集包含一个变量集合和若干组对应的变量值，用于模拟字段名向量 F 和字段值向量的集合$\{X\}$。然后，使用这些数据集构造相应的数据库事务，并在事务的执行过程中使用原型工具进行不变式推导，如果得到的结果与数据集中包含的不变式一致，则认为不变式的推导结果准确。此外，实验还将结果与 Daikon 推导的结果进行横向对比，以分析原型工具的结果是否准确和简明。

Daikon 的不变式推导包括在源代码插入跟踪点、执行测试套件获取运行数据、对数据集进行分析等一系列流程，其主要的分析对象是代码中的类和方法。为了使 Daikon 间接地支持对数据库事务进行分析，实验基于需要分析的数据集的相关数据，模拟构造 Daikon 运行过程的中间文件，包括定义文件(declaration files)、跟踪文件(trace files)等，使其在形式和结构上与 Daikon 跟踪源代码后产生的数据记录一致[16]。然后，将这些中间文件输入 Daikon 的不变式分析模块，以实现利用 Daikon 对数据库事务中的数据进行推导。

由于该过程并没有对 Daikon 的源代码和算法做出本质性的改动，只是间接地调用了其中的部分模块，因此可以认为该实验方法具有公平性和有效性。

例如，对于数据库事务语句：INSERT INTO person (name,sex,age) values("Bob","M",22)，其生成的跟踪文件如图 6 所示。

```
ppt Test. listField()::ENTER
ppt-type enter
variable name
variable-kind variable
…
variable sex
variable variable
…
variable age
variable variable
…
Test. listField()::ENTER
this_invocation_nonce
1
name
"Bob"
1
sex
"M"
1
age
22
1
```

图 6 构造的跟踪文件的核心内容

5.2 不变式推导的质量

本节的目标是通过实验来分析原型工具的不变式推导的质量，即对原型工具的不变式推导的准确率进行统计和分析。

实验共使用了两种数据集。第一种是正确数据集，其中每一组数据都是一个变量元组$(x_1,x_2,\cdots,x_n)$，并且它们全部遵循一个特定的代数方程$\sum_{i=1}^{n}a_i x^{e_i}=0$($a_i$，$e_i$ 为常量)，其中 a_i，e_i 为随机生成的常整数；第二种为含有异常数据的数据集，它在正确数据集的基础上引入了少量不符合其中代数方程的数据作为干扰。

5.2.1 节的实验目标是验证原型工具能否准确地推导出利用正确数据集构造的数据库事务中存在的不变式，同时将结果与 Daikon 推导的不变式在形式和数量上进行对比，以分析原型工具的结果是否准确和简明。在 5.2.2 节的实验中使用含有异常数据的数据集，其中的异常数据本身并不满足事务本身存在的不变式，它模拟的是在运行时以小概率出现的异常数据，以分析原型工具对异常数据的分析准确度。

5.2.1 对正确数据集的实验

该实验使用正确的测试数据集进行实验，其中每一个数据集中包含一个不变式和若干组完全符合该不变式的取值样本。一组正确的数据集如表 1 所列。

表 1 一组正确的测试数据集

不变式：$4x^2+y+z-20=0$		
x	y	z
1	5	11
0.5	30	−11
−3	0	−16
…	…	…

实验使用 50 组数据集进行测试。对于每一组数据集，利用其中所有的取值样本构造若干个数据库事务，然后在运行时使用原型工具和 Daikon 对事务进行不变式分析。由于 Daikon 仅支持对不超过 3 个变量的数据推导形如$z=ax+by+c$(x,y,z 为变量，a,b,c 为常系数)的线性不变式，因此实验中对 Daikon 支持和不支持的数据集分别进行统计和分析，得到的结果如表 2 和表 3 所列。

表 2 原型工具和 Daikon 对 Daikon 支持的数据集的实验结果

	原型工具	Daikon
准确推导的不变式数目/组	16	19
准确推导的不变式比例/%	76.2	90.5

表 3 原型工具对 Daikon 不支持的数据集的实验结果

	原型工具	Daikon
准确推导的不变式数目/组	20	—
准确推导的不变式比例/%	69.0	—

基于以上结果，对原型工具的不变式推导质量分析如下：

(1)对于 Daikon 支持的具有形如 $z=ax+by+c$ 的线性等式不变式的数据样本，Daikon 推导出其中存在的不变式的准确率比原型工具更高，其可能的主要原因是 Daikon 受浮点误差的影响较小。例如，对于不变式为 $5x+3y-7z+40=0$ 的数据样本，原型工具在构造方程组和进行求解的过程中会积累浮点误差，导致最后推导出的不变式为 $1.67x+y-2.34z=0$，或者无法推导出不变式。而由于 Daikon 支持对线性不等式形式的不变式进行推导，如 $5x+3y-7z+40\leqslant 0$，这使得某些数据样本的浮点误差在 Daikon 中不会明显地表现出来。

(2)对于 Daikon 不支持的具有形如 $a_1x_1^{k1}+a_2x_2^{k2}+\cdots+a_nx_n^{kn}+c=0$ 的代数等式不变式的数据样本，原型工具对其进行不变式推导的准确率比对具有线性等式不变式的数据样本更低，可能的主要原因也是受方程求解过程中浮点误差的影响。由于数据在进行高次幂的乘方运算时会增加浮点数的计算量，导致浮点误差增大，从而降低不变式推导的准确率。

5.2.2 对含有异常数据的数据集的实验

本节中的数据集是在 5.2.1 节的正确数据集的基础上引入了少量不符合事务不变式的异常数据，用于模拟运行时以较小概率出现的数据异常。一组含有异常数据的测试数据集如表 4 所列(其中加粗的数据表示异常数据)。

表 4 一组含有异常数据的测试数据集

不变式：$4x^2+y+z-20=0$		
x	y	z
1	5	11
0.5	30	−11
−3	0	−16
2	**8**	**7**
…	…	…

实验使用 50 组数据集进行测试。在加入异常数据之前，首先使用原型工具对这 50 组数据集构造的数据库事务进行不变式推导的测试，确保在仅包含正确数据的前提下原型工具能准确推导出每一组数据集中的不变式。然后，以一定比例向每一组数据中混入异常数据样本，再使用原型工具进行不变式推导。调整异常数据的比例进行多次实验，对不变式推导的准确率取平均值，结果如表 5 所列。

表 5　原型工具对含有异常数据的数据集的实验结果

	准确推导的不变式数目/组	准确推导的不变式比例/%
0	50	100
1%	50	100
3%	49	98
5%	48	96
10%	41	82
15%	37	74
20%	33	66
25%	27	54

基于以上实验结果可以得出：当异常数据的比例较低(小于 5%)时，原型工具对不变式的推导结果几乎不受影响，主要原因是在推导过程中算法从大量的数据样本中随机选取数据进行多次推导，以降低偶然性和排除异常数据。虽然在异常数据比例较高(大于 10%)时，原型工具推导不变式的准确率出现了明显下降，但是在实际的软件运行过程中一旦数据出现如此高的异常率，应当判定为系统的编码设计或运行状态出现了严重问题，此时系统已不满足 3.1 节中进行不变式分析的前提假设。

5.3　不变式推导的性能

本节的目标是通过实验来分析原型工具推导不变式的性能。实验使用的计算机的硬件配置和系统环境如表 6 所列。

表 6　性能测试使用的计算硬件配置和系统环境

硬件和系统环境	配置
处理器	Intel(R) Core(TM) i5-3210M @ 2.50 GHz
内存	8 GB
硬盘	750 GB
操作系统	Windows 7 Ultimate SP1 64-Bit

实验通过调整数值变量数目和度进行推导，对原型工具的所有模块的运行时间之和取平均值，结果如表 7 所列。

表 7　性能测试的实验结果/ms

$\lvert X\rvert$ \ d	1	2	3	4
2	28	32	40	50
4	33	41	51	68
6	35	44	60	72
8	36	46	68	88
10	39	50	72	97

基于以上实验数据可知，随着数值变量数目 $|X|$ 和度 d 的增大，运行时间会随着线性方程组求解复杂度的增加而增加，但其实际值(小于 100ms)均在可以接受的合理范围之内。同时，在推导线性等式形式的不变式时原型工具的用时几乎可以忽略不计，这不会影响软件系统本身的运行效率。

结束语　本文对基于数据库事务的代数等式形式的不变式推导进行了研究，提出了构造并求解线性方程组的推导算法，实现了基于运行时数据的不变式推导的原型工具。原型工具通过捕获软件运行时的数据库事务中的 SQL 语句，对其中的字段及其取值进行提取，以推导出其中存在的不变式。利用推导出的不变式，可以挖掘软件的缺陷和漏洞，以及监控运行时数据的一致性。未来对原型工具的改进主要包含以下两个方面：1)考虑扩展工具的推导类型，使其支持更多类型的不变式的推导；2)考虑改进工具的推导质量，提高不变式推导的准确率。

参 考 文 献

[1] WANG H Z. Quality management of big data: problems and progress [J]. Science & Technology Review, 2014(34): 78-84. (in Chinese)
王宏志. 大数据质量管理：问题与研究进展[J]. 科技导报，2014(34)：78-84.

[2] PERKINS J H, KIM S, LARSEN S, et al. Automatically patching errors in deployed software[C]// Proceedings of the ACM SIGOPS 22nd Symposium on Operating systems Principles. ACM, 2009: 87-102.

[3] 吴世忠. 软件漏洞分析技术[M]. 北京：科学出版社，2014：26-28.

[4] MEYER B. Design by Contract[M]. Prentice Hall, 2002.

[5] MEYER B. Eiffel: The Language[M]. Prentice-Hall, Inc., 1992: 20-30.

[6] LEAVENS G T, BAKER A L, RUBY C. Preliminary Design of JML: A behavioral interface specification language for Java[J]. ACM SIGSOFT Software Engineering Notes, 2006, 31(3): 1-38.

[7] SCHILLER T W, DONOHUE K, COWARD F, et al. Case studies and tools for contract specifications[C]// Proceedings of the 36th International Conference on Software Engineering. ACM, 2014: 596-607.

[8] ERNST M D, COCKRELL J, GRISWOLD W G, et al. Dynamically discovering likely program invariants to support program evolution[J]. IEEE Transactions on Software Engineering, 2001, 27(2): 99-123.

[9] LEMIEUX C. Mining temporal properties of data invariants [C]// 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE). IEEE, 2015: 751-753.

[10] HANGAL S, LAM M S. Tracking down software bugs using automatic anomaly detection[C]// Proceedings of the 24th International Conference on Software Engineering. ACM, 2002: 291-301.

[11] ERNST M D, PERKINS J H, GUO P J, et al. The Daikon system for dynamic detection of likely invariants[J]. Science of Computer Programming, 2007, 69(1): 35-45.

[12] 金路. 高等数学(第 3 版上)[M]. 北京：高等教育出版社，2008.

[13] NGUYEN T V, KAPUR D, WEIMER W, et al. Using dynamic analysis to discover polynomial and array invariants[C]// 2012 34th International Conference on Software Engineering (ICSE). IEEE, 2012: 683-693.

[14] P6Spy[OL]. https://github.com/p6spy.

[15] 蔡雪焘. Java 开发利器：Hibermate 开发及整合应用大全(珍藏版)[M]. 北京：清华大学出版社，2006：579-583.

[16] COBB J, JONES J A, KAPFHAMMER G M, et al. Dynamic invariant detection for relational databases[C]// Proceedings of the Ninth International Workshop on Dynamic Analysis. ACM, 2011: 12-17.