非函数依赖程序不变量动态检测技术研究

刘树锟[1],陈继锋[1,2],阳小华[3]
LIU Shu-kun[1],CHEN Ji-feng[1,2],YANG Xiao-hua[3]

1.湖南涉外经济学院 计算机科学与技术学部,长沙 410205
2.西安交通大学 计算机软件研究所,西安 710049
3.南华大学 计算机科学与技术学院,湖南 衡阳 421001
1.Department of Computer Science and Technology,Hunan International Economics University,Changsha 410205,China
2.Institute of Computer Software,Xi'an Jiao tong University,Xi'an 710049,China
3.Department of Computer Science and Technology,University of South China,Hengyang,Hunan 421001,China
E-mail:llsskkllsskk@163.com

LIU Shu-kun,CHEN Ji-feng,YANG Xiao-hua.Research of dynamical detecting technique of non-functional dependence program invariant.Computer Engineering and Applications,2008,44(35):158-162.

Abstract: In this paper,the notation of program invariant based on the contract is described,and a theory model of dynamically generating technique of program invariant is researched and built.In the theory model,the theory method and technology of dynamically generating technique of program invariant of non-functional dependence are discussed.A new method of dynamically generating technique of program invariant of non-functional dependence based on the theory of database is proposed.Then,a series of detecting measures of specific non-functional dependence program invariants are described.The method detecting program invariants in the way of SQL query,so many kinds of program invariants can be dynamically discovered by the means.User can detect the program invariants on the condition of their interest by giving their query conditions at will.Comparing with Daikon,the method has two excellent features.Firstly,it is better expansible based on the RDBMS technology;Secondly,discovering of program invariants can be accomplished by the function of SQL query based on the technology of relational database which can detect the program invariants in a nimble way.

Key words: program invariant; non-functional dependence; dynamically detect;quality of software;condition query

摘 要:讨论了程序不变量的内涵,研究并建立了程序不变量动态生成系统的理论模型。在该模型中,针对非函数依赖程序不变量动态生成理论、方法和技术进行了阐述。基于数据库的理论提出了一种新的非函数依赖程序不变量动态检测技术,针对各种常见非函数依赖程序不变量类型建立了一系列检测方法。此不变量检测技术通过数据库中提供的SQL条件查询功能,灵活地检测各种常见类型的非函数依赖程序不变量,并且可以根据用户的实际需要随时指定新的程序不变量查询条件。该方法和Daikon等现有的程序不变量检测工具检测方法比较具有明显的特色和优势:第一,基于关系数据库技术,具有良好的可扩展性;第二,使用SQL条件查询功能实现非函数依赖程序不变量检测,检测方法具有很好的灵活性。

关键词:程序不变量;非函数依赖;动态检测;软件质量;条件查询
DOI:10.3778/j.issn.1002-8331.2008.35.048 **文章编号**:1002-8331(2008)35-0158-05 **文献标识码**:A **中图分类号**:TP311

迄今为止软件已经成为最广泛、最重要的应用系统之一,从而软件的质量成为人们目前广泛关注、高度重视的热点问题之一。众所周知,基于合约的程序设计是提高软件质量的一种重要的技术[1],但是合约的形式化制定是一项比较艰巨的任务。然而经过研究发现,程序不变量的动态生成技术是解决合约形式化制定的一项有效方法[2]。

目前关于程序行为描述的生成方法已经有了大量研究,这方面的工作主要可以分为两种类型——静态分析和动态分析。静态分析方法通过检查程序代码等文档发现系统隐含的性质,从而产生相应的形式化规格说明,也就是合约。例如,通过人工检查.NET中ArrayList类的Metadata[2-3],找出一系列隐含的性质,包括类不变量、过程的前提和后置条件(所谓不变量(Invariant)就是一些用于描述在程序运行时保持不变的性质的逻辑断言)。文献[4]中实现了在文档、构造函数、接口实现和基类

基金项目:湖南省自然科学基金(the Natural Science Foundation of Hunan Province of China under Grant No.05JJ30117);湖南省教育厅基金资助课题(No.08C516);湖南省教育厅重点基金资助课题(No.07A034)。
作者简介:刘树锟(1979-),男,硕士,主要研究领域为软件工程、数据库技术;阳小华(1963-),男,教授,博士,博士生导师,主要研究领域为软件工程、数据库技术、信息检索技术等;陈继锋(1966-),男,博士,副教授,主要研究领域为软件工程。
收稿日期:2008-07-07 **修回日期**:2008-11-12

中找寻 Java 类和 EJB 中隐含的性质。在一定的条件下静态分析方法确实能够发现程序中所隐含的重要性质,但是这类技术通常只能成功应用于中小规模的程序。因此,通过程序的实际运行发现其行为合约的动态分析方法近年来得到了长足的发展[5-6],其中影响最大的是麻省理工学院 Ernst 等人的系列研究工作。Ernst 等开发了一个动态发现程序不变量(program invariants)的工具 Daikon[7-9]。Daikon 的不变量都是关于数据行为的不变量,每个不变量中变量个数限制在 3 个以下(含 3 个)。Daikon 主要检查单个变量是否恒为常量、是否恒不为 0、是否恒满足某个函数(如偶数 odd、求模 mod 等)、是否局限在较小的集合或区间内等, 检查两个或三个变量之间是否有相等关系、不等关系、次序关系、线性关系等。

Daikon 已经成功应用于程序演进与重构[8-9]、程序测试与排错[10]、辅助定理证明[11]、构件升级替换[12]等。由于 Daikon 成功的影响,Hangal 等开发了一个动态不变量发现与检测引擎 DIDUCE[13](Dynamic Invariant Detection U Checking Engine)。DIDUCE 被成功地应用于 JSSE(Java Secure Socket Extension)等较大规模程序的排错,从而例证了动态不变量发现与检测方法对于大型程序的有效性。Daikon、DIDUCE 等工具虽然取得了很大的成功,但仅仅是程序行为合约(不变量)生成技术研究的开端。

深受上述研究的启发,为程序不变量检测技术提出了一个理论模型。根据关系数据库理论把程序不变量分为非函数依赖和函数依赖两种类型,主要介绍运用数据库技术如何检测非函数依赖程序不变量的方法。通过建立的理论模型不仅可以检测一些直观的不变量形式,还可以根据用户实际需要定义新的不变量表达式。这样在一定程度上解决了 Daikon、DIDUCE 等工具只能检测一些直观的不变量形式,而不能灵活的定义新的不变量表达式的问题。

1 程序不变量动态生成技术理论模型

在此建立了基于合约的程序不变量动态检测技术的理论模型。定义为一个形如<P,L,N,V,T,A,Ins,Tre,Ext>的九元组。其中,P 是程序集合,L 是程序中位置的集合,N 是程序中变量名的集合,V 是程序中各种类型变量值域的集合,T 是测试用例集合,A 是程序不变量集合;$Ins:P->P\times L\times N$ 称为程序编配映射;$Tre:P\times L\times N\times T->P\times L\times N\times V$ 称为程序轨迹映射;$Ext:P\times L\times N\times V->P\times L\times A$ 称为断言生成映射。

程序不变量匹配检测分为函数依赖程序不变量与非函数依赖程序不变量两种基本类型。函数依赖程序不变量是指那些存在某种函数关系的变量之间的规则,可以包含变量间的线性关系,以及其它函数关系等等。这些程序不变量可以是二元的、三元的也可以是多元的不变量。相对与函数依赖而言,非函数依赖范围比较广泛了,所有不是函数依赖的关系统称为非函数依赖关系,例如变量间大小关系、次序关系、不等关系等。

2 非函数依赖程序不变量基本类型

程序中变量与变量之间存在的特定关系以及变量自身具有的某些性质的重要性是毋庸置疑的。这些重要性质主要可以表现为对数字变量关系的描述, 也可以表现为非数字变量之间,序列变量、集合变量之间等关系描述。而这些关系主要可以分为大小关系、次序关系、不等关系、时序关系、范围关系等等。这些关系都属于简单非函数依赖不变量的范畴。简单的非函数依赖程序不变量形式基本如下所示(其中 x、y、z 是变量,a、b、c 是常量)。

2.1 针对一个单一的数字变量的不变量

(1)确定此数字变量的范围,例如:是否属于小集合:指出变量仅具有一个小数量的不同值。例如 $x\in\{a,b,c\}$,其中 a、b、c 为常数。

范围界限:$x\geqslant a$,$x\leqslant b$ 和 $a\leqslant x\leqslant b$,指出变量的最小值或最大值。

(2)是否具有某种不等关系,例如:非常量:x 不等于某个常量,指出变量从来都不是某个常量即 $x\neq c$(c 为常量)。

非函数关系:x 不满足于某个具体的函数表达式,指出 x 不满足某个具体的函数表达式即 $x\neq f(y)$。

2.2 针对一个单一的非数字变量的不变量

(1)确定此变量的范围,例如:是否属于小集合:指出变量仅具有一个小数量的不同的值.例如 $x\in\{a,b,c\}$,其中 a、b、c 为常量。

(2)是否具有某种不等关系,例如:非常量:x 不等于某个常量,指出变量从来都不是某个常量即 $x\neq c$(c 为常量)。

非函数关系:x 不等于某个具体的函数表达式,指出 x 不满足某个具体的函数表达式即 $x\neq f(y)$。

2.3 针对一个单一的序列变量的不变量

(1)确定序列变量的范围,例如:按字典顺序确定最小和最大的序列值(例如指出字符串或数组值的范围)。

(2)确定此序列的增减性,例如:指出序列的元素是不增的,不减的或相等的。

2.4 针对两个序列变量的不变量

确定序列间的某种关系用 $R(x,y)$ 表示,其中 x、y 代表两个不同的序列。R 关系可以包含子序列关系(例如 x 是 y 的子序列)、反向关系(例如 x 是 y 的反向)、不等关系(例如 $x<y$,$x\leqslant y$,$x>y$,$x\geqslant y$,$x\neq y$ 等)。上述的程序不变量的类型是非函数依赖程序不变量中比较典型、常见类型。当然用户也可以根据实际需要来检测符合用户指定的类型的程序不变量。

3 程序运行轨迹的存储结构

程序可以认为是初始状态变换到最终状态的算法描述,变换过程为程序的执行。所以,要分析程序内部各个部分的内在联系以及隐含的程序不变量,必须要分析程序执行过程中的各个状态变换。随着程序的运行,大量的数据轨迹生成了,要分析程序的运行状态,必须把程序的实时运行轨迹保存下来。在此以 Java 程序为例,说明轨迹文件的存储结构。从而可以利用数据库理论与技术分析程序的运行轨迹。所以数据的存储模式是至关重要的。在介绍存储格式之前,首先了解一下 Java 程序的结构。Java 程序的结构,如图 1 所示。

程序动态运行的过程实质上可以认为是状态转换的过程。由上面程序代码可以看出 Java 程序主要是一个层次结构即程序包→类→方法→变量。所以,在分析程序的过程中就是在分析上述层次结构的具体信息。因此在记录动态状态信息时候也要按照层次结构收集。

由 Java 程序结构图可以看出程序的层次结构。所以,如果按照程序的结构层次来记录程序的运行状态信息,对于将来程序内部各个层次关系的分析是很有利的。在数据库中定义类关

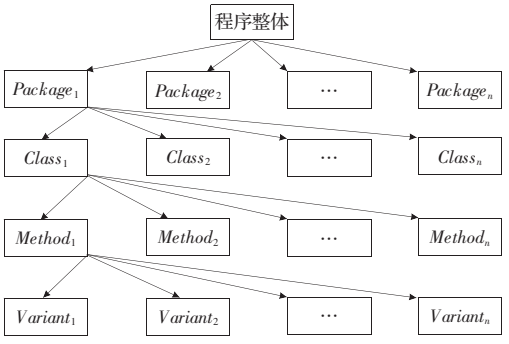

图 1 Java 程序结构图

系(关系名称为程序中每一个类的名称)、方法关系(关系名称为程序中每一个方法的名称)、变量关系(关系名称为程序中每一个变量的名称)。数据库中的存储关系定义如下：在数据库中,以一个八元组的形式分别记录表示轨迹存储格式文件的 8 个关系。该八元组定义为:<*RC*,*RM*,*RIGNC*,*RJGC*,*RMLNC*,*RNLC*,*RPDNC*,*RQDC*>,其中 *RC* 是类描述关系,*RM* 类代表各个方法关系,*RIGNC* 代表全局非集合变量关系,*RJGC* 全局集合变量,*RMLNC* 定义局部非集合变量,*RNLC* 定义局部集合变量,*RPDNC* 定义域非集合变量,*RQDC* 定义域集合变量。对于程序中的数组以及其它集合类型的处理,采用的是主表关联子表的方式。如上面描述的存储格式中,针对数组类型的变量,处理方式是把所有相关的数组名称存储于一张数据表中,每一个数组中所有的值存储在一张数据表中,通过数组名称把两张表关联在一起。这种方法适用于所有的集合类型变量。

4 非函数依赖程序不变量检测

针对上文提到的非函数依赖程序不变量的各种体现形式,可以用存储在数据库中的轨迹数据作为不变量提取的数据基础。非函数依赖程序不变量的检测基本上是基于数据库技术的,即以关系数据库管理系统(DBMS)的 SQL 查询功能为基础加上高级程序语言编程技术来求解。

4.1 非函数依赖程序不变量提取方法

在程序不变量动态检测技术的主要理论模型的基础上,现在提出非函数依赖程序不变量检测的方法模型:可以简单地定义为一个形如<*CON*,*SENPOOL*,*INVS*>的三元组,其中:*CON* 是非函数依赖程序不变量检测条件集合,*SENPOOL* 是为不变量检测得语句模版集合,*INVS* 是检测到的非函数依赖程序不变量的集合。

所有的非函数依赖程序不变量的检测条件都要保存在上述定义的三元组<*CON*,*SENPOOL*,*INVS*>的 *CON* 中,根据条件集合中的检测条件,会生成针对这些条件的 SQL 语句模版。生成的这些语句模版会存在于 *SENPOOL* 中。在非函数依赖程序不变量检测的过程中，则根据所要检测的具体的不变量类型,基于上述三元组中的 *SENPOOL* 中的 SQL 条件语句模版,依照准确的检测条件生成具体的查询语句,查询出满足条件的非函数依赖程序不变量,把这些不变量保存于 *INVS* 中,最后展现给用户。用户在检测的过程中也可以自己指定查询条件,这些查询条件也会保存在 *CON* 中，这样随着查询条件会不断的增多,所能查询的非函数依赖程序不变量也会不断增多。因为条件集合中的条件元素增多了,那么模版也会随之增多,从而检测到的程序不变量的覆盖范围更加广泛。因此,本文的程序不变量检测方法也可以称为对用户开放型的不变量检测方法。

4.2 单一的数字变量的不变量检测

针对上面所提及的单一的数字变量的不变量的几种典型类型的非函数依赖程序不变量可以作下面的处理。

比如,要判断 x 是否在一个小集合内,只需要按照数据表中列 x 的值对表进行分组查询,若只有一组,则有不变量 $x=c$(c 为某一常量);若只有很少的几组,则有不变量 $x\in\{a,b,c\cdots\}$。可以通过下列语句实现:

SELECT x FROM Table group by x

要判断是否有不变量 $x\neq 0$,则只需以 $x\neq 0$ 为条件作一次查询即可。即:

SELECT x FROM Table where x! =0

要判断此变量的范围只需要计算出最大和最小值即可:

取最大值:SELECT MAX(x) FROM Table

取最小值:SELECT MIN(x) FROM Table

则 x 的范围为 MAX(x)>x>MIN(x)。

要判断是否有不变量 $x!\equiv$某一函数表达式,则只需以 $x!\equiv$某一函数表达式为条件,作一次条件查询就可解决。例如检测 $x!\equiv a(\mathrm{mod}\ b)$,则只要用下面的 SQL 语句查询即可:

SELECT x FROM Table where x! =a(mod b)

一般而言,要判断单个数字变量 x 是否有性质 $f(x)$,只要以 $f(x)$为条件对观测点数据表进行查询,然后将查询结果与原数据表对比大小即可。即 SELECT x FROM Table where 条件表达式。

综上所述,单一数字变量的不变量检测的算法基本流程可以定义如下:(1)检测该变量是不是在一个小的集合范围内,如果有满足条件的则保存临时检测结果;(2)检测该变量是否有最大、最小值,如果有满足条件的则保存临时检测结果;(3)检测该变量是否不满足某个具体的函数关系 $f(x)$,若有满足条件的则保存临时检测结果;(4)如果存在检测到的程序不变量,则输出所有的程序不变量结果。

4.3 单一的非数字变量的不变量检测

针对上面所提及的单一的非数字变量的不变量的几种典型类型的非函数依赖程序不变量可以作下面的处理。比如,要判断 x 是否在一个小集合内,只需要按照数据表中列 x 的值对表进行分组查询,若只有一组,则有不变量 $x=c$(c 为某一常量);若只有很少的几组,则有不变量 $x\in\{a,b,c,\cdots\}$:

SELECT x FROM Table group by x

要判断是否有不变量 $x!\equiv$某一函数表达式,则只需以 $x!\equiv$某一函数表达式为条件,作一次条件查询就可解决。例如检测 $x!\equiv a(\mathrm{mod}\ b)$,则只要用下面的 SQL 语句查询即可:

SELECT x FROM Table where x! =a(mod b)

一般而言,要判断单个非数字变量 x 是否有性质 $f(x)$,只要以 $f(x)$为条件对观测点数据表进行查询,然后将查询结果与原数据表对比大小即可。即 SELECT x FROM Table where 条件表达式。

综上所述,单一非数字变量的不变量检测的算法基本流程可以定义如下:(1)检测该变量是不是在一个小的集合范围内,有则临时保存;(2)检测该变量是否不满足某个具体的函数关系 $f(x)$,如果有则临时保存;(3)如果有检测到的程序不变量则输出所有的检测的不变量结果。

4.4 两个数字变量的不变量检测

要判断是否有不变量 $x<y$、$x\leqslant y$、$x>y$、$x\geqslant y$、$x=y$、$x\neq y$,则只

需分别以 $x<y$、$x\leqslant y$、$x>y$、$x\geqslant y$、$x=y$、$x\neq y$ 为条件进行查询,若查询结果与原表一样,则不变量 $x<y$,$x\leqslant y$,$x>y$,$x\geqslant y$,$x=y$,$x\neq y$ 成立,反之则不成立。

通用的查询语句为:

SELECT　变量列表　FROM Table where 条件表达式(变量间的关系)

综上所述,两个数字变量的不变量检测的算法基本流程可以定义如下:(1)检测是否满足不变量 $x\neq y$,如果满足条件则临时保存结果进入(2),否则直接进入(3);(2)检测是否满足不变量 $x<y$,如果满足则保存临时结果,进入(6)否则进入(4);(3)检测是否满足不变量 $x\leqslant y$,如果满足条件则进入(6);(4)检测是否满足不变量 $x>y$,如果满足则临时保存结果进入(5),否则进入(6);(5)检测是否满足不变量 $x\geqslant y$,如果满足条件则临时保存结果,进入(6);(6)输出所有的检测到的满足条件的非函数依赖程序不变量结果。

这种方法不仅对简单数据类型十分有效,而且对涉及复杂数据类型的不变量也是可行的,只要构造相对复杂的嵌套查询即可。

4.5　单一的数字序列变量的不变量检测

对于序列(x)的范围不变量的确定,只需利用查询语句出此序列(x)或者数组的最大值和最小值即可。对应的 SQL 语句如下:

SELECT MAX(x) FROM Table

SELECT MIN(x) FROM Table

那么 x 的范围也就是最大值和最小值之间。

针对序列变量进行范围不变量检测的算法流程:

(1)求出最大值(SELECT MAX(x) FROM Table);

(2)求出最小值(SELECT MIN(x) FROM Table);

(3)确定变量范围;

(4)输出结果。

对于单一序列中元素单调性不变量判断,判断序列中所有元素具有相等性,只需要语句 SELECT MAX(x) FROM Table 和语句 SELECT MIN(x) FROM Table 如果最大值和最小值是相等的,那么就可以判断该序列是元素相等的。

相等性检测算法流程:

(1)求出最大值(SELECT MAX(x) FROM Table);

(2)求出最小值(SELECT MIN(x) FROM Table);

(3)比较两者(最大值和最小值)是否相等;

(4)如果相等则报告序列具有元素相等性输出不变量信息,否则退出变量相等性检测。

判断序列元素是否具有增减性:

语句 SELECT x FROM Table order by x DESC 和 SELECT x FROM Table order by x ASC 可以按照递减和递增两种顺序排列 x 列。

判断是否递减:

是否递减检测方法如下所示:

按照语句 SELECT x FROM Table order by x DESC 得到一个有序序列,把这个序列存入临时表中。然后分别定义两个游标,依此比较原序列和新序列的每一个值(通过游标定位),如果两两相等那么原序列有递减性;如果比较过程中有不等情

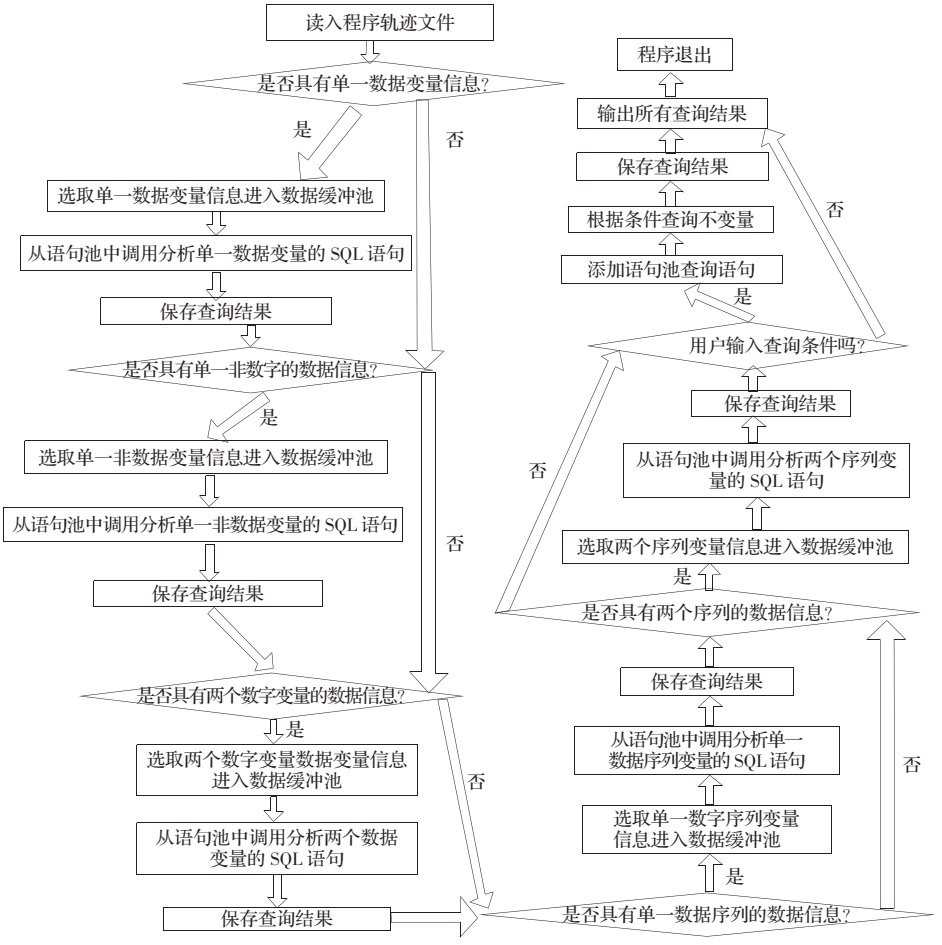

图 2　不变量检测流程图

况出现,那么退出比较。从而判断该序列没有递减性。

是否递增检测方法如下所示:

语句 SELECT x FROM Table order by x ASC 得到一个有序序列,把这个序列存入临时表中。然后分别定义两个游标,依此比较原序列和新序列的每一个值(通过游标定位),如果两两相等那么原序列有递增性;如果比较过程中有不等情况出现,那么退出比较。从而判断该序列没有递增性。如果既不满足递减也不满足递增那么该序列无序。

4.6 两个序列变量的不变量检测

序列的反向判断(例如 x 是 y 的反序列):

首先判断两个序列是否具有相同的元素数目,通过语句(1)、(2)判断:

(1)SELECT count(*) FROM *Table*1;

(2)SELECT count(*) FROM *Table*2。

序列的反向判断检测算法:

如果返回结果相同,那么定义两个游标分别指向 *Table*1中 x 的首元素和 *Table*2 中 y 的尾元素,比较两个元素值是否相同。如果相等那么第一个游标增加 1,第二个游标减 1,再次两两比较游标指向的元素值,游标分别加 1 和减 1,直到第一游标达到最大值,结束比较过程。如果比较过程中有两元素不等情况出现,则结束比较过程。从而判断两个序列没有反序关系;否则认为具有反序列关系。

子序列判断(例如 x 是 y 的子序列)检测算法:

首先计算各自的元组数目:

A=SELECT count(*) FROM *Table*1

B=SELECT count(*) FROM *Table*2

如果 $A<B$,则定义 *table*1 中指向 x 元素的游标(*flagx*)值为 1,从 B 首位置依此选择 A 个元组与 *Table*1 比较,如果同则为子序列;否则继续从 *Table*2 中的第 $A+1$ 位置截取 A 个元素比较,直到(B–*flagx*)的初始值小于 A 比较过程结束。

序列比较(例如 $x<y, x\leqslant y, x>y, x\geqslant y, x\neq y$ 等)

序列大小比较检测算法:比较序列含有元素的多少可以借助语句 SELECT count(*)FROM Table 计算每一个序列含有的元素个数。然后比较得到的各自的元组数目大小即可。

4.7 非函数依赖程序不变量检测算法总体流程

在检测程序不变量的整体过程中采用开放性质的语句池机制。这种机制的优点是,SQL 语句池是开放的。将事先定义好典型性质的不变量的检测 SQL 语句存储在语句池中,当进行轨迹文件分析时,可以从语句池中调用 SQL 语句。如果没有用户感兴趣的不变量查询条件,那么用户可以根据自己的意愿,随时构造 SQL 语句然后存入语句池中(因为语句池是对用户开放的,用户可以随时把 SQL 语句存入语句池中)。这样用户就可以根据自己添加的查询条件进行非函数依赖程序不变量提取。其具体检测流程如图 2 所示。而且本文建立了程序不变量分析器的原型系统。运行情况如图 3 所示。程序实际运行效果表明,提出的基于数据库的非函数依赖程序不变量检测算法不仅是可行的,而且是非常有效的。

5 小结

建立了程序不变量检测的理论模型,对具体的简单非函数依赖程序不变量类型提出的一系列的检测方法形式灵活多变,可以根据用户的需要,临时根据指定的程序不变量形式进行检测,这是一种新的程序不变量的检测方法。不仅可以挖掘出多种类型的不变量信息,而且此不变量分析方法打破了 Daikon、Diduce 等程序不变量检测工具的检测形式固定化、单一化的局面(Daikon、Diduce 等工具只针对一元、二元、三元不变量进行检测)。其明显的特色和优势在于:第一,基于关系数据库技术,具有良好的可扩展性;第二,使用 SQL 条件查询功能实现非函数依赖程序不变量检测,检测方法具有很好的灵活性。

图 3 不变量分析器运行图

参考文献:

[1] Mitchell R,Mckim J,Meng Y.Design by Contract 原则与实践[M].北京:人民邮电出版社,2003.

[2] Polikarpova N,Meyer B.A comparative study of programmer-written and automatically inferred contracts,ETH Zurich Technical Report608[R].Switzerland,2008.

[3] Yang J,Evans D.Perracotta: Mining temporal API rules from imperfect traces[C]//ICSE,2006:282-291.

[4] Saff D,Artzi S,Perkins J H,et al.Automatic test factoring for Java[C]//Proceedings of the 21st Annual International Conference on Automated Software Engineering,Long Beach,CA,USA,2005:114-123.

[5] Braberman V,Fernández F.Parametric prediction of heap memory requirements[C]//International Symposium on Memory Management,Tucson,AZ,USA,2008:141-150.

[6] Lorenzoli D,Mariani L,Pezzè M.Automatic generation of software behavioral models[C]//Proceedings of the 30th International Conference on Software Engineering,Leipzig,Germany,2008:501-510.

[7] Papi M M,Ernst M D.Practical pluggable types for Java[C]//Proceedings of the 2008 International Symposium on Software Testing and Analysis,Seattle,WA,USA,2008:201-212.

[8] Ernst M D.Dynamically discovering likely program invariants[D].Seattle,Washington:University of Washington Department of Computer Science and Engineering,2000.

[9] Ernst M D,Perkins J H,Guo P J.The daikon system for dynamic detection of likely invariants[J].Science of Computer Programming,2007,69(3):35-45.

[10] Ren X X,Chesley O C.Identifying failure causes in Java programs:an application of change impact analysis[J].IEEE Transactions on Software Engineering,2006,32(9):718-732.

[11] Win T N,Ernst M D.Using simulated execution in verifying distributed algorithms[J].Software Tools for Technology Transfer,2006,6(1):67-76.

[12] Mariani L,Pezzè M.Dynamic detection of COTS component incompatibility[J].IEEE Software,2007,24(5):76-85.

[13] Hangal S,Lam M S.Tracking down software bugs using automatic detection[C]//Proceedings of the 24th International Conference on Software Engineering,2002:291-301.