微函数依赖及其推理

孙纪舟[1)] 李建中[1)] 高 宏[1)] 刘显敏[2)]

[1)](哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)

[2)](哈尔滨工业大学软件学院 哈尔滨 150001)

摘 要 起初,作为一个数据库模式设计的工具,函数依赖理论得到了很多的关注,而在数据修复中,该理论并不是十分有效.近年来,针对不一致数据的检测和修复问题,更多的约束被提出来,包括条件函数依赖、修复规则以及编辑规则等.然而,这些方法都只关注了属性整体之间的依赖关系,而实际应用中的数据通常有属性部分之间的依赖关系.例如,某单位员工的工号前两位决定了其所属的部门,而此类依赖信息就被已有方法忽略.该文首先提出了一类更一般化的约束——微函数依赖,微函数依赖引入提取函数,用来表示属性的部分信息.利用提取函数之间的依赖关系,能够检测出更多的不一致数据.理论方面,该文首先研究了微函数依赖的可满足性问题和蕴含问题,然后提供了一个正确且完备的推理系统.最后,通过实验证实了微函数依赖能够在可接受的时间开销内检测出更多的错误数据.

关键词 微函数依赖;提取函数;可满足性问题;蕴含问题;推理系统

中图法分类号 TP311 **DOI 号** 10.11897/SP.J.1016.2016.02134

Reasoning About Micro Dependencies

SUN Ji-Zhou[1)] LI Jian-Zhong[1)] GAO Hong[1)] LIU Xian-Min[2)]

[1)](*School of Computer Science and Technology, Harbin Institute of Technology, Harbin* 150001)

[2)](*School of Software, Harbin Institute of Technology, Harbin* 150001)

Abstract Originally, functional dependency theory got a lot of attentions as a schema designing tool, which is not so effective in data repairing. Recent years, more constrains have been proposed to detect and repair inconsistent data, including conditional functional dependencies (CFDs), fixing rules and editing rules, etc. However, to the best of our knowledge, all of the proposals focus on dependencies between entire attributes, while there are ubiquitous dependencies between partial information of the attributes in the real world. For example, the id's 2-length prefix of an employee may determine her department, while this kind of dependencies have been ignored by previous proposals. In this paper, we firstly propose a class of more general constrains, referred to as micro-dependencies(MDs). Extracting functions(EFs) are involved into MDs to extract partial information from attributes. With dependencies between EFs, more inconsistent data in a dataset can be detected. For static analysis of MDs, we then investigate the satisfiability problem and the implication problem analogous to those for CFDs. And then a sound and complete inference system for implication analysis is developed. Finally, we experimentally show that MDs can detect much more errors in data with an acceptable time cost.

Keywords micro-dependency; extracting function; satisfiability problem; implication problem; inference system

收稿日期:2015-10-18;在线出版日期:2016-03-04.本课题得到国家"九七三"重点基础研究发展规划项目基金(2012CB316202)、中央高校基本科研业务费专项资金(HIT.NSRIF.201649)和国家自然科学基金(61502121)资助.**孙纪舟**,男,1985 年生,博士研究生,主要研究领域为数据质量、弱可用海量数据数据上的近似计算.E-mail:sjzh@hit.edu.cn.**李建中**,男,1950 年生,教授,博士生导师,主要研究领域为数据库、海量数据处理、物联网和无线传感器网络等.**高 宏**,女,1966 年生,教授,博士生导师,主要研究领域为无线传感器网络、物联网、海量数据管理和数据挖掘等.**刘显敏**,男,1984 年生,博士,讲师,主要研究方向为海量数据计算、数据质量管理.

1 引　言

相关资料表明，脏数据正在变得越来越普遍且不可避免，例如，在信息产业还没有特别发达的1998年就已经有1%～5%的商业数据存在错误，在有些企业中甚至达到了30%[1]；国际著名科技咨询机构Gartner的调查显示，全球财富1000强企业中超过25%的企业信息系统中的数据不正确或不准确[2]. 这些脏数据通常带来很严重的后果[3]，而且由于数据量庞大，数据问题种类繁多，要想解决数据质量问题带来的严重后果，也是一项消耗巨大的工作. 美国公司每年花在处理脏数据上的资金就高达6千亿美元[4]；目前数据质量工具的市场以每年17%的速度增长，远超IT行业平均增长率7%；一项近期统计显示，在大多数的数据仓库项目中，数据修复工作占开发时间和预算的30%～80%[5]. 近年来，数据库领域广泛研究了处理脏数据的问题. 不一致性是脏数据的最重要方面之一. 如果一个数据库违反了某些数据质量规则如函数依赖[6]、条件函数依赖[7-8]、扩展的条件函数依赖[9]、编辑规则[10]、修复规则[11]等，则称它是不一致的.

起初，检测和修复脏数据的依据是传统的函数依赖(FDs)[6]，而函数依赖主要用于数据库的模式设计，在处理不一致数据时，往往不够充分. 实际上，在一个精心设计① 的数据库中，所有的函数依赖都是主键约束，即约束前件都是候选键. 同时，所有的主流数据库都支持SQL语句中的UNIQUE和KEY关键字，来避免插入和修改数据时违反函数依赖，如例1.

例1. 考虑数据库模式 $Employee(\underline{Eid}, Name, Dept, Position, EntryDate, Salary)$ 以及一个相应的实例 r，如表1所示. 带有下划线的属性 Eid 是主键，由主键定义知，Eid 的值可以确定其他属性的值. 另外，来自同一部门的员工不能重名，因此 $Name$ 和 $Dept$ 也构成了一个候选键. 这两条约束可以形式化为两条函数依赖 $\phi_1: [Eid] \rightarrow [Name, Dept, Position, EntryDate, Salary]$ 和 $\phi_2: [Name, Dept] \rightarrow [Eid, Position, EntryDate, Salary]$.

由于底层数据库管理系统的约束检查机制，r 中的所有元组在 $[Eid]$ 以及在 $[Name, Dept]$ 上的值都不相同(在 $[Eid]$ 或 $[Name, Dept]$ 上具有相同值的元组会被禁止插入数据库)，因此上述两条约束也不会被违反.

从例1可以看到，函数依赖能够阻止不一致数据进入数据库，但是对检测错误是不充分的. 为此，Fan等人对函数依赖进行了扩展，把语义相关的值绑定到约束中去，即条件函数依赖(*CFDs*)[7-8]. 条件函数依赖的主要特点就是它可以只对表中部分满足一定条件的元组起作用，而函数依赖则对整个表中的所有元组起作用. 在例1中，如果每个部门只能有一个管理者，就可以得到以下的条件函数依赖：φ_1：$[position = \text{Manager}, Dept] \rightarrow [Eid, Name, EntryDate, Salary]$ 表示当一个员工是管理者的时候，通过他所在的部门，能够知道他的员工号、姓名等信息. 函数依赖对此是无法表示的.

然而，根据IDEF1X①，如果数据库模式设计得很完善，$Employee$ 将会按照分类属性 $Position$ 被分成表 $Staff(Eid, Name, Dept, EntryDate, Salary)$ 和 $Manager(Eid, Name, Dept, EntryDate, Salary)$. 这样，$\varphi_1$ 就会被等价替换成关系 $Manager$ 上的一个函数依赖 $\phi_3: [Dept] \rightarrow [Eid, Name, EntryDate, Salary]$. 由前述内容可知，$\phi_3$ 不易被违反. 另外，每个条件函数依赖在每个属性上至多绑定一个数值，导致它的表达能力有限. 为了增加其表达能力，Fan等人又提出了内置谓词的条件函数依赖($CFD^p s$)[12]以及扩展的条件函数依赖(*eCFDs*)[9]. 前者把条件函数依赖中的'='扩展到'$\neq$'，'$<$'，'$\leq$'，'$>$'和'$\geq$'，而后者是把条件函数依赖中的"单值条件"扩展到"有限集合条件"，即只有属性值在(或者不在)集合中时才满足条件. 这两种扩展都很大程度上增加了条件函数依赖的表达能力. 然而，$CFD^p s$ 主要针对数值型属性而 *eCFDs* 不能够处理"无穷集合"的情况. 此外，其他研究工作提出来的约束包括编辑规则[10]、修复规则[11]、差分约束[13]和可比较约束[14]等. 但是，所有这些约束都只关注了整个属性值之间的关系，忽略了这样一个事实：两个属性值的部分信息相等也能够确定一些相等关系. 比如，两个电话号码的前几位相同，则它们可能来自同一地区. 而在之前的工作中，完全相同的两个电话号码才被认为来自同一地区. 在例1中，如果 Eid 的长度为2的前缀表示员工所在的部门，可以将其表示为 $[prefix(Eid, 2)] \rightarrow [Dept]$，本文中表1中的第一

① Data Modeling Method (IDEF). http://www.idef.com/idef1x-data-modeling-method/

条和最后一条元组的 *Eid* 前缀相同，表示员工来自相同部门，但是他们的 *Dept* 属性又不相同，说明数据不一致，而之前的工作都没有检测出这类不一致.

表 1 Employee 关系的一个实例 *r*

Eid	*Name*	*Dept*	*Position*	*EntryDate*	*Salary*/%
01M00001	Alice	R&D	Manager	2007-09-01	8000
01S00002	Bob	R&D	Staff	2012-03-12	2000
02M00001	Carol	Test	Manager	2009-08-29	6500
02S00010	David	Test	Staff	2013-09-01	3000
03M00001	Eric	O&M	Manager	2012-08-25	6000
03S00002	Frank	O&M	Staff	2012-08-31	2000
01S00003	Gary	O&M	Staff	2014-03-01	2000

针对这些实际情况，本文的主要贡献包括：

(1) 提出微函数依赖(*MDs*)的概念，它是对 *FDs*、*CFDs*、*CFD*^{p}*s* 和 *eCFDs* 的进一步扩展，以在数据修复中发现更全面的不一致数据. 这里之所以用“微函数依赖”而不用“部分函数依赖”的名称，是因为后者已经在关系数据库的函数依赖理论[6]中被占用，并且表示了完全不同的意思. 通过引入提取函数，来提取属性中的部分信息，*MDs* 能够检测出更多的不一致数据. 这部分内容主要在第 2 节介绍；

(2) 研究了关于 *MDs* 的静态分析问题，包括两部分，分别是可满足性问题和蕴含问题. 可满足性问题是指，给定一个微函数依赖集 Σ(根据惯例，Σ 在本文中表示依赖集，并非求和符号)，判断是否存在一个满足 Σ 的非空实例. 蕴含问题是指，给定一个微函数依赖集 Σ 和单条微函数依赖 φ，判断“任一实例满足 Σ”是否蕴含“该实例也满足 φ”，即判断：满足 Σ 的实例是否也一定满足 φ. 这部分内容主要在第 3 节中介绍.

(3) 针对 *MDs*，给出了若干推理规则，构成一个推理系统，并且证明了该推理系统的正确性和完备性. 这部分内容主要在第 4 节介绍.

(4) 介绍了用微函数依赖检测数据错误的算法，首先给出基于 SQL 查询的方法，然后给出基于排序的方法，并在多条依赖的情况下对其进行优化. 这部分内容在第 5 节进行介绍.

(5) 进行了充分的实验，实验结果表明 *MDs* 确实能够比之前方法检测出更多的错误数据. 这部分内容主要在第 6 节介绍.

最后，第 7 节介绍了相关工作，第 8 节对本文工作进行了总结.

2 微函数依赖

R 是一个关系模式，$\{A_1, A_2, \cdots, A_n\}$ 是 R 上的属性集合，记作 $attr(R)$. 对于 $attr(R)$ 中的每个属性 A_i，都有相应的域 $dom(A_i)$. R 上的一个实例 r 是指一个有限的元组集合：$r \subseteq dom(A_1) \times \cdots \times dom(A_n)$. 元组 t 和属性 A 构成一个元组-属性对，表示 t 在 A 上的值，称之为单元格，记作 $t[A]$.

函数依赖是形如 $X \rightarrow Y$ 的规则，其中 X, Y 均为属性集合，其语义为 R 上的实例 r 应满足：对于 r 中的任意两条元组 s, t，如果 $s[X]=t[X]$，则应该有 $s[Y]=t[Y]$. 在下文中，将称“$\rightarrow$”之前的部分(即 X)为约束前件，之后的部分(即 Y)为约束后件.

条件函数依赖是在函数依赖的基础上，增加了常量条件，比如 $[A=a, B] \rightarrow C$ 表示，如果 r 中有两条元组在 A 属性上的值相等且都为 a，在 B 属性上值也相等，那么这两条元组在 C 上也应相等，否则说 r 不满足该条约束.

下面，将介绍提取函数的概念，并定义两种类型的微函数依赖：用户定义的微函数依赖和固有微函数依赖.

2.1 提取函数

为了表示部分属性之间的关系，本文引入提取函数的概念. 对 $attr(R)$ 中的每个属性 A，都有一个相应的提取函数集 $F^A=\{f_1^A, f_2^A, \cdots, f_l^A, i^A\}$，其中每个提取函数都可以看成是由属性 A 得到的新的属性，称之为 A 上的导出属性，其属性值由相应元组、函数语义以及属性 A 共同确定：元组 t 在 f_i^A 上的值为 $f_i^A(t[A])$，为了和基本属性的表示方法相统一，本文将 $f_i^A(t[A])$ 记为 $t[f_i^A]$. 需要注意的是，F^A 中的每个提取函数 f 都定义在 A 的域的子集上，即 $dom(f) \subseteq dom(A)$. 特别的，i^A 是 $dom(A)$ 上的恒等函数，也就是 $i^A=A$，即对于所有的 $a \in dom(A)$，$i^A(a)=a$. 如果 $a \notin dom(f)$，$f(a)$ 上无定义，用空值来表示此语义：$f(a)=\text{null}$. 不失一般性，对于两个不同的属性 A 和 B，可以认为 $F^A \cap F^B=\varnothing$. 将 R 中涉及到的所有提取函数记为 $\mathcal{F}$，即 $\mathcal{F}=\bigcup_{i=1}^{n} F^{A_i}$.

简便起见，在不引起歧义的前提下，可以省略上下标. 对于一个提取函数 $f \in F^A$，其相应的属性 A 被记作 $Att(f)$. 为了与主流数据库中函数的用法相一致，本文允许函数的参数为一个属性，或者具体的值. 当函数参数是一个属性的时候，其输出是一个新

的属性，即导出属性；当函数参数是一个具体值的时候，函数输出也是一个具体值. 例如 SQL 中有字符串函数 *reverse*(*str*)，表示把 *str* 中的所有字符反序. 例如，在表 1 中所示的数据库上执行查询语句：

Q1：Select *Eid*, *Name*, *reverse*(*Name*) From Employee；

得到的结果如表 2 所示.

表 2　在 Employee 上执行 Q1 得到的结果

Eid	*Name*	*reverse*(*Name*)
01M00001	Alice	ecilA
01S00002	Bob	boB
02M00001	Carol	loraC
02S00010	David	divaD
03M00001	Eric	cirE
03S00002	Frank	knarF
01S00003	Gary	yraG

查询结果仍然是一个关系，*reverse*(*Name*)是其中一个属性，而每条元组在 *reverse*(*Name*)属性上对应的值，等于 *Name* 属性上的值取反转操作得到. 例如 *reverse*(*Name*)在第 1 条元组上的值为 *reverse*(Alice)，即 ecilA.

实际上，每个提取函数 f 都从相应的属性中提取了一部分信息，它可以被看做一个新的导出属性. 微函数依赖就是定义在这些导出属性上，从而表示部分属性之间的关系.

2.2　微函数依赖

和函数依赖相比，微函数依赖的主要区别在于它定义在提取函数上，而不是直接定义在属性上. 类似于函数依赖，一个微函数依赖 φ 具有如下的形式：

$$[f_1, f_2, \cdots, f_k] \rightarrow f_0,$$

其中 $f_i \in \mathcal{F}, i=0,1,\cdots,k$. 显然，传统的函数依赖是只有恒等函数的微函数依赖. 给定 R 上的关系实例 r，如果对于 r 中的任意两条元组 t_1, t_2 都满足：

(1) $t_1[f_1, f_2, \cdots, f_k] \neq t_2[f_1, f_2, \cdots, f_k]$；或

(2) $t_1[f_0] = t_2[f_0]$.

则称 r 关于 φ 一致，或 r 满足 φ，记作 $r \models \varphi$；否则称 r 关于 φ 不一致，或 r 不满足 φ，记作 $r \not\models \varphi$. 对于微函数依赖集合 Σ 来说，如果对 $\forall \varphi \in \Sigma$，$r \models \varphi$ 都成立，则称 r 关于 Σ 一致，或 r 满足 Σ，记作 $r \models \Sigma$；否则称 r 关于 Σ 不一致，或 r 不满足 Σ，记作 $r \not\models \Sigma$.

根据语义的不同，微函数依赖被分为两类：用户定义的微函数依赖（*UMDs*）和固有微函数依赖（*IMDs*）. 前者是由用户定义的，可能会被违反，而后者是由提取函数的定义决定的，永远不会被违反.

2.2.1　用户定义的微函数依赖

顾名思义，此类依赖是指用户希望数据库应该满足的微函数依赖. 在例 1 中，*Eid* 的前两位用来识别员工所在部门 *Dept*，接下来两位表示入职日期 *EntryDate*，第 5 位表示职位 *Position*. 这些规则可以分别用微函数依赖表示为 $[prefix(Eid,2)] \rightarrow [Dept]$，$[substr(Eid,3,2)] \rightarrow [EntryDate]$ 以及 $[charAt(Eid,5)] \rightarrow [Position]$. $prefix(str,len)$ 函数返回 *str* 串的前 *len* 位，$substr(str,begin,len)$ 函数返回 *str* 串从 *begin* 位置开始且长度为 *len* 的子串，$charAt(str,idx)$ 函数返回 *str* 串第 *idx* 位的字符. 这些约束都是由用户定义的，属于用户定义的微函数依赖，当数据产生错误时，它们有可能被违反. 例 1 中第 1 条元组和最后一条元组的 *Eid* 前两位相同，应该属于同一部门，但两条元组的 *Dept* 属性值不相等，约束 $[prefix(Eid,2)] \rightarrow [Dept]$ 被违反.

另外，函数依赖和条件函数依赖都可以用微函数依赖来表示. 对于函数依赖，只需要把其中所有的属性用相应恒等函数代替即可，只是形式发生了变化，语义未变. 例如 $[Name, Dept] \rightarrow [Eid]$ 可表示为 $[i_1(Name), i_2(Dept)] \rightarrow [i_3(Eid)]$，其中 i_1, i_2, i_3 分别为相应属性上的恒等函数，简便起见，本文将省略这些恒等函数，例如 $[Name, Dept] \rightarrow [Eid]$ 就可以被看作一个合法的微函数依赖.

为表示条件函数依赖，则需要利用空值，并认为“空值不等于任何值”. 例如为了用微函数依赖表示规则 $[Position = \text{Manager}, Dept] \rightarrow [Eid]$，引入提取函数 $isManager(str)$：

$$isManager(str) = \begin{cases} 1, & \text{if } str = \text{Manager} \\ \text{null}, & \text{if } str \neq \text{Manager} \end{cases},$$

等价的微函数依赖就可以写成

$$[isManager(str), Dept] \rightarrow [Eid].$$

如果元组的 *Position* 属性不等于 Manager，约束前件出现空值 null，因为空值不等于任何值，所以该元组和其它元组在约束前件上都不会相等，从而不会违反该约束，使得该约束只在一部分元组上起作用，和对应的条件函数依赖是等价的.

在例 1 中，上述的函数依赖和条件函数依赖都未被违反. 而微函数依赖 $[prefix(Eid,2)] \rightarrow [Dept]$ 被违反，因为 Alice，Bob 和 Gary 的 *Eid* 属性前缀都是 01，他们应该属于同一个部门，但是 Gary 却来自不同的部门 O&M，由此可以看出，微函数依赖能够检测出一些已有方法忽略掉的错误数据.

至于为什么不直接把 $\mathcal{F}$ 中的函数 f 当作新的属

性 f_A,这样一来微函数依赖实际上就变成了普通的函数依赖,本文不这样做的原因有如下几点:首先,属性 A 和 f_A 之间的映射关系由函数 f 的定义确定,如果简单的把 $f(A)$ 看作新的属性,这个映射信息就会丢失;第二,引入新的函数依赖 $\varphi:A\rightarrow f(A)$,并不能彻底地表达 A 和 f_A 之间的紧密关系,因为 f 把 A 映射到一个确定的值上,而 φ 仅仅是说两条元组在属性 A 上相等也应该在 f_A 上相等,而不知道应该等于何值,例如,单条元组可能不会违反 φ,却有可能违反 f 的定义;最后,即使可以用条件函数依赖 $[A=\mathrm{a}]\rightarrow[f_A=f(\mathrm{a})]$ 表达相同的语义,但需要对 $dom(A)$ 中的所有值分别给出条件函数依赖,这是不可行的,因为 $dom(A)$ 可能很大,甚至是无穷的,这时需要用无穷多个条件函数依赖表达相同的语义,是不现实的.

本小节中提到的规则都是用户定义的,被称为用户定义的微函数依赖(*UMDs*).从例子中可知,当数据出现错误时,这些约束可能被违反,根据违反情况,可以检测出数据中的不一致.

2.2.2 固有微函数依赖

考虑单个属性 A,定义在 A 上的提取函数保留了 A 的部分信息,例如字符串属性 A 的前两位子串保留在 $prefix(A,2)$ 中.如果有多个提取函数定义在 A 上,这些函数保留的信息之间可能互不相交,或者会有重叠,甚至相互包含.这就导致了同一个属性上,不同提取函数之间的依赖关系.例如容易得到如下依赖关系:$[i(A)]\rightarrow[prefix(A,4)]$ 以及 $[prefix(A,4)]\rightarrow[prefix(A,2)]$.两条约束分别是说,整个属性的值能够确定该属性值的前缀,长度为 4 的前缀能确定长度为 2 的前缀.对于一个整数属性 B,有 $[mod(B,2),mod(B,3)]\rightarrow[mod(B,6)]$ 等,该约束的含义是,如果整数能被 2 和 3 同时整除,则它也能够被 6 整除.

另外,如果 $f_1,f_2\in F^A$,$dom(f_1)\cap dom(f_2)=\varnothing$,即对于任意元组 t,$t(f_1)$ 和 $t(f_2)$ 至少有一个为空值.为表示这类约束,为每个提取函数 f 引入两个域函数 $\widetilde{f}$ 和 $\overline{f}$,分别指示 f 的定义域及其补集,定义如下:

$$\widetilde{f}(x)=\begin{cases}1, & \text{if } f(x)\neq\text{null}\\ \text{null}, & \text{if } f(x)=\text{null}\end{cases},$$

$$\overline{f}(x)=\begin{cases}\text{null}, & \text{if } f(x)\neq\text{null}\\ 1, & \text{if } f(x)=\text{null}\end{cases}.$$

容易看出这两个域函数总是取不同的值,前面 f_1 和 f_2 的关系就可以表示为 $[\widetilde{f}_1]\rightarrow[\overline{f}_2]$ 且 $[\widetilde{f}_2]\rightarrow[\overline{f}_1]$,表示如果 f_1 和 f_2 有一个不为空值,那么另一个就一定是空值.本文用符号 $\mathcal{DF}$ 表示所有域函数的集合,即 $\mathcal{DF}=\{\widetilde{f},\overline{f}\mid f\in\mathcal{F}\}$.

容易看出,本小节的所有约束都是由函数的定义决定的,与数据无关,也就是说,不管数据的不一致程度有多高,这些约束都不会被违反.由于这些约束的固有特性,本文称之为固有微函数依赖(*IMDs*).和用户定义的微函数依赖不同,固有微函数依赖永远不会被违反,因此并不能够直接用来发现数据中的错误,其主要作用在于对微函数依赖的推理中,具体用法详见第 3、4 节.

值得注意的是,一个提取函数可以定义成数学表达式、字符串表达式或甚至一段程序代码等,自动判断固有微函数依赖的工作变得几乎不可能实现.本文重点在于分析微函数依赖的性质,因此在后续章节中,若无特别说明,都假定已经知道了完备的固有微函数依赖集.为了加以区分,*UMDs* 集合和 *IMDs* 集合被分别标记为 Σ_U 和 Σ_I.

3 微函数依赖的静态分析

微函数依赖的提出,是为了检测出更多的错误数据.为了更好的利用已有微函数依赖集 Σ,有必要判断 Σ 自身是否存在不一致和冗余,即本节将讨论的可满足性分析以及蕴含性分析,统称为静态分析.本节重点研究了可满足性问题和蕴含性问题各自的复杂性.

3.1 微函数依赖的可满足性

给定关系模式 R 上的 Σ_U 以及完备的 Σ_I,微函数依赖的可满足问题是指,判断是否存在一个非空关系实例 r,使得 $r\models\Sigma_U$,记作 MD-SAT 问题.

为便于分析,用 Σ_d 表示把 Σ_U 中涉及到的所有提取函数 f 替换成相应的域函数 $\widetilde{f}$,得到的新依赖集.一个直观的结论就是 Σ_U 的可满足性与其中的提取函数的定义域有关,而与具体函数值无关.

引理 1. Σ_U 可满足当且仅当 Σ_d 可满足.另外,Σ_d 可满足当且仅当存在一个单元组关系实例 r,使得 $r\models\Sigma_d$.

证明. 首先证明 Σ_U 可满足当且仅当 Σ_d 可满足:

$\rightarrow$:需证如果 Σ_U 可满足,则 Σ_d 可满足.因 Σ_U 可满足,一定存在一个关系 r,使得 $r\models\Sigma_U$.只要证明 $r\models\Sigma_d$,就可以得出 Σ_d 可满足.

考虑 r 中的任意两条元组 t_1 和 t_2 以及 Σ_d 中的任意一条约束 $\varphi_d:[\widetilde{f}_1,\widetilde{f}_2,\cdots,\widetilde{f}_l]\rightarrow\widetilde{f}_0$,则 φ_d 在

Σ_U中对应的约束为 $\varphi_U:[f_1,f_2,\cdots,f_l]\rightarrow f_0$. 假设 $t_1[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]=t_2[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]$，因为空值不等于任何值，所以两条元组在 $\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l$ 上的值均不为空. 由 2.2.2 节中对域函数的定义，容易得出元组 t_1 和 t_2 在 $f_1,f_2,\cdots,f_l$ 上的值也都不为空. 所以有 $t_1[f_1,f_2,\cdots,f_l]=t_1[f_1,f_2,\cdots,f_l]$，且 $t_2[f_1,f_2,\cdots,f_l]=t_2[f_1,f_2,\cdots,f_l]$，又因为 $r\models\Sigma_U$，容易得出 $t_1[f_0]=t_1[f_0]$，$t_2[f_0]=t_2[f_0]$，即 t_1 和 t_2 在 f_0 上不为空，进而 t_1 和 t_2 在 $\tilde{f}_0$ 上不为空，根据 2.2.2 节的定义，域函数只有 null 和 1 两种取值，所以 $t_1[\tilde{f}_0]=t_2[\tilde{f}_0]=1$，得出 $r\models\Sigma_d$，进而得出结论：如果 Σ_U 可满足，则 Σ_d 可满足.

←：证如果 Σ_d 可满足，则 Σ_U 可满足. 因 Σ_d 可满足，一定存在一个关系 r，使得 $r\models\Sigma_d$. 只要构造一个关系 r'，并证明 $r'\models\Sigma_U$，就可以得出 Σ_U 可满足的结论.

令 r' 为 r 的一个单元素子集，即 $r'\subseteq r$ 且 $|r'|=1$. 不失一般性，可假设 $r'=\{t\}$. 考虑 Σ_U 中任意一条约束 $\varphi_U:[f_1,f_2,\cdots,f_l]\rightarrow f_0$，如果 $t[f_1,f_2,\cdots,f_l]=t[f_1,f_2,\cdots,f_l]$，则说明 t 在 $f_1,f_2,\cdots,f_l$ 上均不为 null，根据 2.2.2 节中对域函数的定义，进而得出 t 在 $\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l$ 上也都不为 null，所以 t 在 $\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l$ 上的值全为 1. 因此 $t[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]=t[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]$，因 $r\models\Sigma_d$ 且 $t\in r$，有 $t[\tilde{f}_0]=t[\tilde{f}_0]=1$，可知 $t[f_0]$不为 null，因此 $t[f_0]$是一个非空的确定值，即有 $t[f_0]=t[f_0]$，得出 $r'\models\Sigma_U$，进而得出结论：如果 Σ_d 可满足，则 Σ_U 可满足.

然后证明 Σ_d 可满足当且仅当存在一个单元组关系 r，使得 $r\models\Sigma_d$.

如果存在一个单元组关系 r，使得 $r\models\Sigma_d$，可以直接根据可满足性的定义得出结论：Σ_d 可满足.

如果 Σ_d 可满足，则一定存在关系 r'，使得 $r'\models\Sigma_d$，因此对于 r' 中任意一条元组 t 和 Σ_d 中的任意一条约束 $\varphi_d:[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]\rightarrow\tilde{f}_0$，如果 $t[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]=t[\tilde{f}_1,\tilde{f}_2,\cdots,\tilde{f}_l]$，一定有 $t[\tilde{f}_0]=t[\tilde{f}_0]$，因此构造单元组关系 $r=\{t\}$，直接得出 $r\models\Sigma_d$，即存在一个单元组关系 r，使得 $r\models\Sigma_d$.

综上可得，Σ_U 可满足当且仅当 Σ_d 可满足. 另外，Σ_d 可满足当且仅当存在一个单元组关系 r，使得 $r\models\Sigma_d$.　　证毕.

接下来讨论，如果有一个完备的固有微函数依赖集 Σ_I，MD-SAT 问题的复杂性.

定理 1.　给定完备的 Σ_I，MD-SAT 问题是 NP-完全的.

证明.　首先，微函数依赖是对条件函数依赖的一般化，而条件函数依赖的可满足性问题已经被证明是 NP-完全的，因此 MD-SAT 问题是 NP-难的.

然后证明 MD-SAT 属于 NP，为此，给出一个不确定算法生成只有一个虚拟元组 t 的关系 r，猜测 t 在每个域函数 $\bar{f}$ 上的值(1 或 null)，并判断 r 是否满足 Σ_d 和 Σ_I，若满足，则输出“是”，否则输出“否”. 上述工作在多项式时间内完成，并枚举了所有的可能情况，因此 MD-SAT 属于 NP.

综上可得，MD-SAT 是 NP-完全的.　　证毕.

上述证明并没有直接猜测 t 的各属性上的值，因为属性域可能是无穷的，不满足不确定图灵机的要求. 因此对域函数随机赋值，并根据来 Σ_I 验证赋值是否合法. 不难看出，如果 $\mathcal{F}$ 大小固定，所有可能的赋值就有 $2^{|\mathcal{F}|}$ 种，可以在常数时间内枚举完全，直接可得到如下结论.

定理 2.　如果 $\mathcal{F}$ 大小是固定的，则 MD-SAT 问题是 P 问题.

如果一个微函数依赖集合 Σ_U 是不可满足的，有必要给出有效算法来找出 Σ_U 的一个子集 Σ'_U，并使 $|\Sigma'_U|$ 最大化. 本文将该问题称为“微函数依赖的最大可满足子集”问题，记为 MAXMSAT 问题.

由于 MD-SAT 本身是 NP-完全的，属于难解问题，因此 MAXMSAT 问题更加困难，因此需要考虑近似算法. 只要找到一个保留近似因子的归约方法，从 MAXMSAT 归约到已有的可近似问题，就能够说明 MAXMSAT 是可近似的. 本文将其归约到 MAXGSAT 问题，即一般化的极大可满足问题(Maximum Generalized Satisfiability)，而对于该问题，已经有近似算法. MAXGSAT 问题简单描述如下：给定一个布尔表达式的集合 $\Phi=\{\phi_1,\phi_2,\cdots,\phi_n\}$，找出一个变量赋值，使得 Φ 中为真的表达式数量最大. 归约过程包含两个多项式时间可计算的函数 g 和 h，使得对任意一个微函数依赖集合 Σ_U，满足：

(1) $g(\Sigma_U)$ 是 MAXGSAT 问题的一个实例 Φ_Σ，即一个根据 Σ_U 计算得到的一个布尔表达式集合，且

(2) 如果 Φ_m 是 Φ_Σ 的一个可满足子集，则 $h(\Phi_m)$ 是 Σ_U 的一个可满足子集，且

(3) $|OPT_g(g(\Sigma_U))|\geq|OPT_m(\Sigma_U)|$，且

(4) $|h(\Phi_m)|\geq|\Phi_m|$

其中 $OPT_g(g(\Sigma_U))$ 是 MAXGSAT 问题的最优解，$OPT_m(\Sigma_U)$ 是 MAXMSAT 问题的最优解，结合第(3)、(4)条中的不等式，很显然如果 MAXGSAT 有一个近似比为 ε 的算法，通过归约，也能够得到

MAXMSAT 问题的近似比为 ε 的算法.

首先考虑如何根据给定的 Σ_U,以及完备的固有函数依赖集 Σ_I,构造 $g(\Sigma_U)$. 对每一个导出函数 f,引入一个布尔变量 x_f 来表示 f 是否实例化为空值 null. 对 Σ_I 中的每一个微函数依赖 φ,则可以用一个析取式 ϕ_φ 来表示 φ 是否被满足. 如果 $\widetilde{f}$ 出现在"→"的左边,或者 $\bar{f}$ 出现在"→"的右边,则将变量 $\bar{x}_f$ 加入到 ϕ_φ 中. 如果 $\bar{f}$ 出现在"→"的左边,或者 $\widetilde{f}$ 出现在"→"的右边,则将变量 x_f 加入到 ϕ_φ 中. 例如,对于固有的微函数依赖 $\widetilde{f}_1,\widetilde{f}_2,\widetilde{f}_3\rightarrow\widetilde{f}_4$,只要满足下列条件之一,约束即被满足:

(1) f_1 被实例化为空,或

(2) f_2 被实例化为空,或

(3) f_3 被实例化为非空,或

(4) f_4 被实例化为非空.

对应的布尔表达式 ϕ_φ 就是:$\bar{x}_{f_1}\vee\bar{x}_{f_2}\vee x_{f_3}\vee x_{f_4}$. 特别的,固有微函数依赖集 Σ_I 是永远不会被违反的,用布尔表达式

$$\psi_I=\bigwedge_{\varphi\in\Sigma_I}\phi_\varphi$$

来表示 Σ_I 是否被满足. 对于每一个用户定义的微函数依赖 $\varphi\in\Sigma_U$,其构造方式和固有微函数依赖类似,不同之处在于要给整个布尔表达式和 ψ_I 做合取操作,例如有用户定义的微函数依赖 $\widetilde{f}_1,\widetilde{f}_2,\bar{f}_3\rightarrow\widetilde{f}_4$,其对应的布尔表达式 ϕ_φ 为 $(\bar{x}_{f_1}\vee\bar{x}_{f_2}\vee x_{f_3}\vee x_{f_4})\wedge\psi_I$,这样,得到

$$\Phi_\Sigma=g(\Sigma_U)=\{\phi_\varphi\mid\varphi\in\Sigma_U\}$$

h 函数和 g 函数相反,是把布尔表达式映射为微函数依赖,对于一个布尔表达式集合 $\Phi_m\subseteq\Phi_\Sigma$,$h(\Phi_m)$ 定义为集合 $\{\varphi\mid\phi_\varphi\in\Phi_m\}$.

很显然,函数 g 和 h 都能够在多项式时间内计算出. 且通过归约过程可知,$g(\Sigma_U)$ 是 MAXGSAT 问题的一个实例,条件(1)被满足. 如果 $\phi_\varphi\in\Phi_m$ 被某个赋值满足,则在对应的微函数依赖中,要么"→"左边的提取函数不满足要求,要么"→"右边的提取函数满足要求,而且所有固有微函数依赖都被满足,从而使得存在一个元组满足该微函数依赖,条件(2)被满足. 另外,从归约过程可以看出每一条用户定义的微函数依赖都对应一个不同的布尔表达式,二者是一一对应的,且布尔表达式被满足当且仅当对应的微函数依赖被满足,因此 MAXGSAT 和 MAXMSAT 问题的最优解大小是相等的,条件(3)被满足. 同时由于二者的一一对应关系可知 $|h(\Phi_m)|=|\Phi_m|$,条件(4)被满足. 综上可知,上述归约过程是保留近似比的,因此对于 MAXMSAT 问题存在一个多项式时间的近似算法.

3.2 微函数依赖的蕴含性

不同于可满足性分析,蕴含性分析的目的是为了消除冗余. 给定 R 上微函数依赖集 $\Sigma=\Sigma_U\cup\Sigma_I$ 以及一条微函数依赖 φ,其中 Σ_I 是完备的固有函数依赖集,判断 Σ 是否蕴含 φ. Σ 蕴含 φ 当且仅当对于 R 上的任意实例 r,如果 $r\models\Sigma$,一定有 $r\models\varphi$. 记作 $\Sigma\models\varphi$. 记微函数依赖的蕴含问题为 IMP-MD 问题. 为了研究其复杂性类,首先给出如下引理.

引理 2. 给定 Σ 和 φ,存在一个实例 r 使得 $r\models\Sigma$ 且 $r\not\models\varphi$,当且仅当存在一个不超过两条元组的实例 r' 使得 $r'\models\Sigma$ 且 $r'\not\models\varphi$.

证明. 从两个方向分别证明.

←:显然,如果存在一个不超过两元组的实例 r' 满足 $r'\models\Sigma$ 且 $r'\not\models\varphi$,直接令 $r=r'$,立即得到 $r\models\Sigma$ 且 $r\not\models\varphi$.

→:假设存在一个实例 r 满足 $r\models\Sigma$ 且 $r\not\models\varphi$,因为 $r\not\models\varphi$,由 $\not\models$ 符号的含义可知 r 中一定存在两条元组 t_1,t_2 共同违反 φ,即 $\{t_1,t_2\}\not\models\varphi$. 另外,因为 $r\models\Sigma$,不难得出 r 的任意一个子集仍然满足 Σ,所以 $\{t_1,t_2\}\models\Sigma$. 直接令 $r'=\{t_1,t_2\}$,此部分得证. 证毕.

因此,要判断 $\Sigma\models\varphi$ 是否成立,只需要考虑所有不超过两条元组的实例即可.

定理 3. IMP-MD 问题是 CoNP-完全的.

证明. 由于已知微函数依赖是对条件函数依赖的一般化,而且条件函数依赖的蕴含问题是 CoNP-完全的,可直接得出 IMP-MD 问题是 CoNP-难的.

接下来证明 IMP-MD 问题属于 CoNP,给出一个不确定算法生成有两条虚拟元组 t_1,t_2 的关系 r',对每个提取函数 f,算法猜测 t_1,t_2 在 f 取值的不同组合:

(1) $t_1(f)=t_2(f)$,或

(2) $t_1(f)\neq t_2(f)$ 且二者都不为空值,或

(3) $t_1(f)\neq$null 且 $t_2(f)=$null,或

(4) $t_1(f)=$null 且 $t_2(f)\neq$null,或

(5) $t_1(f)=$null 且 $t_2(f)=$null.

赋值结束后,算法分别判断 r' 是否满足 Σ 和 φ,如果在不确定算法的所有猜测路径上都有:

(1) $r'\not\models\Sigma$,或

(2) $r'\models\varphi$.

则输出"是",否则输出"否".

上述过程在多项式时间内枚举了所有可能组合,所以 IMP-MD 属于 CoNP.

综上可得，IMP-MD 是 CoNP-完全的.　证毕.

不难看出，如果 $\mathcal{F}$ 大小固定，所有可能的赋值组合就有 $5^{|\mathcal{F}|}$ 种，可以在常数时间内枚举完全，直接可得到如下结论.

定理 4.　如果 $\mathcal{F}$ 大小是固定的，则 IMP-MD 问题是 P 问题.

4　微函数依赖的公理系统

Armstrong 公理是函数依赖中蕴含分析的基础工具. 类似的，本文给出微函数依赖的推理系统，记为 $\mathcal{I}$，如表 3 所示. 给定一个微函数依赖集 Σ 和一条微函数依赖 φ，如果能够从 Σ 根据 $\mathcal{I}$ 推出 φ，则标记为 $\Sigma\vdash_I\varphi$.

表 3　微函数依赖的推理系统 $\mathcal{I}$

规则名	内容
规则 1.	如果 $f\in F$，则 $F\to f$.
规则 2.	如果 $F\to f_1, F\to f_2, \cdots, F\to f_k$，且 $[f_1, f_2, \cdots, f_k]\to f$，则 $F\to f$.
规则 3.	如果 $[F,g]\to\widetilde{f}$，则 $[F,\widetilde{g}]\to\widetilde{f}$，类似的如果 $[F,g]\to\overline{f}$，则 $[F,\overline{g}]\to\overline{f}$
规则 4.	如果 $\Sigma\vdash_{\mathcal{I}}([F,\widetilde{g}]\to f)$，而且 $\Sigma\vdash_{\mathcal{I}}([F,\overline{g}]\to f)$，则有 $\Sigma\vdash_{\mathcal{I}}([F,Att(g)]\to f)$.
规则 5.	$[\widetilde{f},\overline{f}]\to g$，其中 f,g 任意.

规则 1 和 2 分别对应 Armstrong 公理中的自反律和传递律. 规则 3 是指约束后件是一个域函数，那么约束前件中的函数都能简化为域函数. 规则 4 是指，在 g 空与非空的情况下，F 都能够决定 f，那么 F 可以和 g 对应的属性共同决定 f. 规则 5 是指，如果约束前件中有矛盾，那么它可以决定任意函数. 接下来讨论 $\mathcal{I}$ 的正确性和完备性.

定理 5.　微函数依赖的推理系统 $\mathcal{I}$ 是正确的. 即若有 $\Sigma\vdash_{\mathcal{I}}\varphi$，则有 $\Sigma\models\varphi$.

证明.　$\mathcal{I}$ 是正确的，当且仅当 $\mathcal{I}$ 中的每条推理规则都是正确的，因此只需分别证明各条推理规则的正确性.

规则 1. 对于 r 中任意两条元组 s,t，如果 $s[F]=t[F]$，已知 $f\in F$，一定有 $s[f]=t[f]$. 得出 $r\models(F\to f)$.

规则 2. 对任一实例 r，如果 $r\models\{F\to f_1,\cdots,F\to f_k,[f_1,\cdots,f_k]\to f\}$，需证明 $r\models(F\to f)$. 对 r 中的任意两条元组 s 和 t，如果 $s[F]=t[F]$，则有 $s[f_i]=t[f_i], i=1,2,\cdots,k$. 即 $s[f_1,f_2,\cdots,f_k]=t[f_1,f_2,\cdots,f_k]$，从而可以得出 $s[f]=t[f]$. 得证 $r\models(F\to f)$.

规则 3. 对于任意实例 $r\models([F,g]\to\widetilde{f})$ 以及 r 中任意元组 t 使得 $t[F,\widetilde{g}]$ 不含空值，一定有 $t[\widetilde{f}]$ 不为空值，否则 $t[F,g]=t[F,g]$ 而 $t[\widetilde{f}]\neq t[\widetilde{f}]$，与 $r\models([F,g]\to\widetilde{f})$ 矛盾. 由 $\widetilde{f}$ 的定义可知 $t[\widetilde{f}]$ 的值总是 1. 因此 r 中任意两条元组，若在 $[F,\widetilde{g}]$ 上相等，也必然在 $\widetilde{f}$ 上相等且都等于 1. 对于 $\overline{f}$ 的情况证明过程类似.

规则 4. 根据域函数的定义可知，$\widetilde{f}$ 和 $\overline{f}$ 互补，二者有且仅有一个等于 1. 对任意实例 r. 如果 $r\models\{[F,\widetilde{g}]\to f,[F,\overline{g}]\to f\}$. 对于 r 中任意两条元组 t,s. 如果 $t[F,Att(g)]=s[F,Att(g)]$，有 $t[\widetilde{g}]=s[\widetilde{g}]=1$ 或 $t[\overline{g}]=s[\overline{g}]=1$. 如果是第 1 种情况，可由 r 满足 $[F,\widetilde{g}]\to f$ 得出 $t[f]=s[f]$，否则，可由 r 满足 $[F,\overline{g}]\to f$ 得出 $t[f]=s[f]$. 得证 $r\models([F,Att(g)]\to f)$.

规则 5. 实例 r 中的任何两条元组 s,t 在 $[\widetilde{f},\overline{f}]$ 上都不会相等，因为 $\widetilde{f},\overline{f}$ 中必有一个为空值，因此依赖 $[\widetilde{f},\overline{f}]\to g$ 永远不会被违反.

综上，*MDs* 的推理系统 $\mathcal{I}$ 是正确的.　证毕.

在证明完备性之前，先定义合法集的概念，如果一个域函数集合 S 满足：对于任意 $f\in\mathcal{F}$，$\widetilde{f},\overline{f}$ 有且仅有一个属于 S，则称 S 是合法的，记所有合法集的集合为 VC. 给定一个函数集合 W，VC 在 W 上的投影记为

$$proj(VC,W)=\{S-W\mid S\in VC\}$$

表示已知 W 中的值全部取非空值时，剩余的所有域函数所构成的全部可能赋值.

算法 1.　m-CLOSURE 算法.

输入：一个微函数依赖集 Σ 和一个函数集 F.

输出：F 关于 Σ 的闭包 $\Sigma_m^*(F)$

1. $result:=F$;
2. REPEAT
3. $\quad cf^+:=\mathcal{F}\cap\bigcap_{vc\in proj(VC,result)}closure(\Sigma,result\cup vc)$;
4. $\quad cd^+:=\mathcal{DF}\cap\bigcap_{vc\in proj(VC,\mathcal{F})}closure(\Sigma,result\cup vc)$
5. $\quad result:=result\cup cf^+\cup cd^+$
6. $\quad$ UNTIL *result* 不再发生变化
7. RETURN *result*;

过程 1.　*closure* 过程.

输入：一个微函数依赖集 Σ 和一个函数集 F.

输出：F 关于 Σ 的单步闭包

1. $result:=F$;
2. IF *result* 包含冲突 THEN
3. $\quad result:=\mathcal{F}\cup\mathcal{DF}$;
4. ELSE
5. $\quad$ REPEAT
6. $\quad\quad$ IF $(G\to g)\in\Sigma$ AND g 不是域函数 THEN

7. IF $G\subseteq result$ THEN
8. $result:=result\cup\{g\}$;
9. END IF
10. ELSE IF $(G\rightarrow g)\in\Sigma$ AND g 是域函数 THEN
11. IF 对 G 中的每个 h 都有 $\tilde{h}\in result$ THEN
12. $result:=result\cup\{g\}$;
13. END IF
14. END IF
15. UNTIL result 不再发生变化
16. END IF
17. RETURN $result$;

定理 6. 微函数依赖的推理系统 $\mathcal{I}$ 是完备的. 即若有 $\Sigma\models\varphi$,则有 $\Sigma\vdash_I\varphi$.

证明. 类似于函数依赖,首先给出一个算法 m-CLOSURE,以 Σ 和 F 为输入,输出一个提取函数集,记为 $\Sigma_m^*(F)$. 表示根据 Σ,F 可决定的所有函数的集合(称为 F 的闭包). 算法 1 的第 3,4 行是枚举域函数的所有可能取值,并把闭包集合求交. 是为了模拟规则 4,不断把符合规则 4 里的 f 添加到闭包里. 注意这里枚举的数量是指数级的,虽然效率很低,但是该算法只是为了辅助证明,并不真是为了计算闭包而设计的.

为证明完备性,先证明如果 $\Sigma\models\varphi$,那么 $f\in\Sigma_m^*(F)$,再证明如果 $f\in\Sigma_m^*(F)$,那么 $\Sigma\vdash_{\mathcal{I}}\varphi$. 其中 φ 等于 $F\rightarrow f$.

对于第 1 步,等价于证明其逆反命题:如果 $f\notin\Sigma_m^*(F)$,那么 $\Sigma\not\models\varphi$. 考虑一个实例 $r=\{s,t\}$,使得对于所有的 $g\in\Sigma_m^*(F)$ 有 $s[g]=t[g]$,且对于所有的 $g'\notin\Sigma_m^*(F)$ 有 $s[g']\neq t[g']$. 对于每个微函数依赖 $G\rightarrow g\in\Sigma$,如果 $G\in\Sigma_m^*(F)$,由 m-CLOSURE 可知 g 也会被加入到 $\Sigma_m^*(F)$ 中,因此 r 满足 Σ 中的所有微函数依赖,即 $r\models\Sigma$. 同时,算法 m-CLOSURE 的初始化步骤保证了 $F\subseteq\Sigma_m^*(F)$,并且根据已知有 $f\notin\Sigma_m^*(F)$,得出 $r\not\models\varphi$ 进而 $\Sigma\not\models\varphi$.

对于第 2 步,首先证明对于所有的 $f\in closure(\Sigma,F)$,$\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$. 一个函数 f 只会在过程 1 的第 1,3,8 和 12 行加入到 $closure(\Sigma,F)$ 中. 如果是在第 1 行加入,规则 1 可以保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$;如果是第 3 行,规则 5 保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$;如果是第 8 行,规则 2 可保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$;如果是第 12 行,规则 2 和 3 可保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$.

接下来,证明对于所有的 $f\in\Sigma_m^*(F)$,$\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$. 这里简单介绍大概证明思路. 一个函数 f 只会在算法 1 的第 1,3,4 行进入 $\Sigma_m^*(F)$. 如果是在第 1 行,规则 1 可以保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$;如果是在第 3 行,递归的使用规则 4 可以将被加入到 $result$ 中的函数被推出,并最终保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$. 如果是在第 4 行,递归的使用规则 3 和 4 将加入到 $result$ 的域函数被推出,并最终保证 $\Sigma\vdash_{\mathcal{I}}(F\rightarrow f)$. 证毕.

5 不一致数据检测

数据清洗的第一步,就是快速有效的检测出数据中的不一致. 本节简要介绍检测数据违反微函数依赖的技术.

给定 R 上的一个关系实例 r,一个微函数依赖集 Σ,需要检测出 r 中违反 Σ 的所有元组. 本节首先讨论如何利用 SQL 查询检测不一致元组,然后给出更有效的、基于排序的检测方法.

5.1 基于 SQL 查询的方法

给定一个微函数依赖 $\varphi:[f_1,f_2,\cdots,f_l]\rightarrow f_0$,可以通过 SQL 语句 Q1 选择出所有违反 φ 的元组:

```
Q1:SELECT f1, f2, ..., fl
   FROM r
   GROUP BY f1, f2, ..., fl
   HAVING COUNT(DISTINCT f0) >1;
```

例如在 *Employee* 关系中,*Eid* 的前两位决定员工所在部门,即 $prefix(Eid,2)\rightarrow dept$,查询语句就可以写成:

```
Q2:SELECT prefix(Eid,2) as Prefix,
          COUNT(DISTINCT dept) as Cnt
   FROM Employee
   GROUP BY prefix(Eid,2)
   HAVING COUNT(DISTINCT dept)>1;
```

Employee 关系中第 1、2、7 条元组的两位前缀都相同,聚集到同一个组里,而且 3 条元组在 *dept* 属性上有 2 个不同值:R&D 和 O&M. 这 3 条元组被检测出来.

5.2 基于排序的方法

SQL 语句检测不一致的方法简单易行,但其缺点是需要先将数据导入到数据库中. 而且上述语句只能找到违反约束的元组,并不能够准确定位到是元组上的哪个属性值违反了约束.

本文提出基于排序的方法,很容易地避免了这些问题. 其大致思想是把所有的元组按照约束前件排序,在约束前件上相等的元组被聚集到了一起,然后检查这些在前件上相等的元组在约束后件上是否全都相等.

利用排序的方法还有一个好处,就是在检测多

条约束的时候，在一定条件下可以进行约束之间共享排序操作的优化. 假设有关系实例 $r=\{t_1,t_2,\cdots,t_n\}$，用户定义的微函数依赖 $\varphi_1:f_1\rightarrow f_1'$ 和 $\varphi_2:f_2\rightarrow f_2'$，固有微函数依赖 $\varphi_I:f_1\rightarrow f_2$. 利用 f_1 和 f_2 之间的依赖关系，只需要进行一次排序，就可以完成对 $\varphi_1:f_1\rightarrow f_1'$ 和 $\varphi_2:f_2\rightarrow f_2'$ 的检测.

定理 7.　已知 R 上有固有函数依赖 $\varphi_I:f_1\rightarrow f_2$，对于 R 上的任意一个实例 r，存在对 r 中所有元组的一种排序，使得在 f_1 和 f_2 上值相等的元组都连续.

证明.　只需要构造一种排序，使得 r 满足在 f_1 和 f_2 上值相等的元组都相邻：以 f_2 为第 1 关键字，f_1 为第 2 关键字，将所有元组从小到大排序. 因为 f_2 是第一排序关键字，因此在 f_2 上相等的所有元组一定是连续的.

另外，对于 r 中任意两条在 f_1 上值相等的两条元组 t_1 和 t_2，排序之后的元组如下所示：

	f_1	f_2
…	…	…
t_1	a_1	b_1
…	…	…
t'	a'	b'
…	…	…
t_2	a_2	b_2
…	…	…

因为 t_1 和 t_2 在 f_1 上的值相等，即 $a_1=a_2$，由固有微函数依赖 $\varphi_I:f_1\rightarrow f_2$ 可知 t_1 和 t_2 在 f_2 上的值也相等，即 $b_1=b_2$. 又由于 f_2 是第 1 排序关键字，所以对任意位于 t_1 和 t_2 之间的元组 t' 来说，其在 f_2 上的值也和 t_1 和 t_2 相等，即 $b'=b_1=b_2$，因此 t_1 到 t_2 之间的所有元组在 f_2 上值相等. 同时因为 f_1 是第 2 排序关键字，所以 t_1 到 t_2 之间的所有元组在 f_1 上是有序的. 由 $a_1=a_2$ 可知，$a'=a_1=a_2$，即证明了在 f_1 上值相等的所有元组，也都是连续的.　　证毕.

元组在 f_1 和 f_2 有序之后，只需一遍顺序扫描，就能检测出违反 $\varphi_1:f_1\rightarrow f_1'$ 或 $\varphi_2:f_2\rightarrow f_2'$ 的所有元组：对于相邻元组 t_1 和 t_2，如果它们在 f_1（或 f_2）上值相等，只需要判断它们在 f_1'（或 f_2'）上的值是否也相等，如果不等，则输出在 f_1（或 f_2）上值等于 $t_1[f_1]$（或 $t_1[f_2]$）的所有元组.

6　实验结果及分析

本文在真实数据上进行试验，验证微函数依赖与函数依赖以及条件函数依赖相比在检测数据错误时的覆盖率以及性能.

6.1　实验环境和数据集

实验环境：英特尔酷睿双核 E7500@2.93 GHz 处理器，金士顿 DDR3 2 GB 1333 MHz 内存，日立 320 GB 7200 转硬盘. 采用 Microsoft Windows 7 操作系统，开发环境为 Microsoft Visual Studio 2008.

实验数据集采用一个扩展的 *Employee* 表（见表 1），包括 10 万条元组，8 个属性：*Eid*（员工编号），*Name*（员工姓名），*Dept*（所在部门），*Position*（职位），*EntryDate*（入职日期），*Salary*（薪水），*Phone*（电话号码）和 *City*（所在城市）.

在 *Employee* 上，有 10 条微函数依赖 $\varphi_0\sim\varphi_9$，其中前 3 条是函数依赖，φ_3 是条件函数依赖，如图 1 所示. 函数 *SubStr*(*str*,*idx*,*len*) 用来从 *str* 中的 *idx* 位置开始，提取长度为 *len* 的子串. *AreaCode* 用来从电话号码中提取区号. *LessThanOneYear*(*date*) 是判断日期 *date* 距今是否小于一年. 所有的这些函数都能够在常数时间内计算出来. $\varphi_4\sim\varphi_7$ 表示 *Eid* 不同位置的子串具有不同的含义. φ_8 表示电话号码的部分信息（区号）可以决定员工所在城市. φ_9 表示所有工作不满一年的员工只能获得相同的基本工资. 为了进行错误检测，首先往 *Employee* 表中引入错误，并得到脏数据表 *errEmp*. 对每一个单元格，其值以概率 *ep*（5%～10%）发生改变，这种改变可以是用表中已有值替换，也可以随机变化（字符串的插入删除或替换字符操作，数值的增加误差操作），两种方式各占一半的概率.

φ_0: [*Eid*] → [*Name*, *Dept*, *Position*, *EntryDate*, *Salary*, *Phone*, *City*];
φ_1: [*Dept*, *City*, *Name*] → [*Eid*, *Position*, *EntryDate*, *Salary*, *Phone*];
φ_2: [*Phone*] → [*City*];
φ_3: [*IsManeger*(*Position*), *Dept*, *City*] → [*Eid*, *Name*, *EntryDate*, *Salary*, *Phone*];
φ_4: [*Substr*(*Eid*,1,2)] → [*City*];
φ_5: [*Substr*(*Eid*,3,2)] → [*EntryDate*];
φ_6: [*Substr*(*Eid*,5,2)] → [*Dept*];
φ_7: [*Substr*(*Eid*,7,1)] → [*Position*];
φ_8: [*AreaCode*(*Phone*)] → [*City*];
φ_9: [*LessThanOneYear*(*EntryDate*)] → [*Salary*];

图 1　关系 *Employee* 上的依赖

6.2　覆盖率

errEmp 中发生错误的单元格集合记作 *ERRORs*. 如果几个单元格共同违反了一个约束 φ，称这几个单元格被约束 φ 检测出来，φ 能够检测出来的所有单元格记为 $Detected_\varphi$，则 φ 的覆盖率为

$$Coverage_\varphi=|Detected_\varphi\cap ERRORs|/|ERRORs|.$$

类似的，对于一个依赖集 Σ：

$$Coverage_{\Sigma}=\frac{|\bigcup_{\varphi\in\Sigma}Detected_{\varphi}\cap ERRORs|}{|ERRORs|}.$$

经过 10 次独立的运行，得到函数依赖、条件函数依赖和微函数依赖各自覆盖率平均值，结果如图 2、图 3 所示，可以看出微函数依赖可以检测出多得多的错误. 注意在本文中，$FDs\subseteq CFDs\subseteq MDs$，意味着 $\varphi_4\sim\varphi_9$ 帮助检测出了两倍多的错误数据. 注意到函数依赖和条件函数依赖覆盖率的曲线几乎是重合的，这是因为二者只相差一个 φ_3，而违反该约束的元组很少.

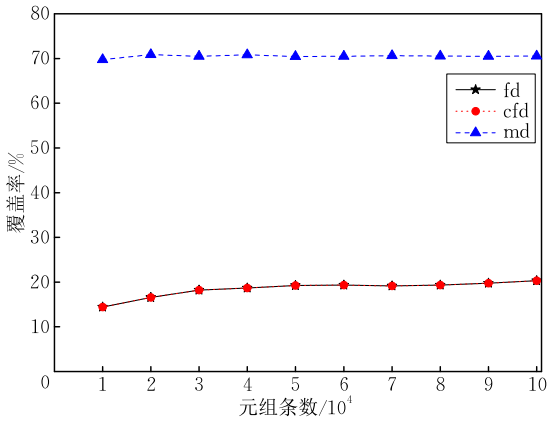

图 2 覆盖率随元组数的变化，错误率＝5%

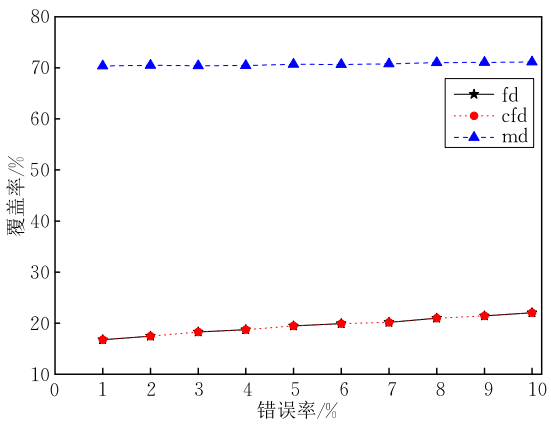

图 3 覆盖率随出错概率的变化，元组条数＝5×10^4

6.3 性 能

为检测错误数据，本文采用基于排序的方法检测约束被违反的情况. 对比了不同类型约束下的时间开销，并对比了优化前后的时间效率.

给定一个微函数依赖 φ，首先以 φ 的约束前件为排序键，并检查相邻的所有元组. 不难得出此方法能够正确地检测出所有不一致数据，且复杂度为 $O(n\log(n))$，其中 n 是元组条数. 图 4 和图 5 描述了每检测一条约束所花的平均时间. 可以看出微函数依赖比函数依赖的代价稍高，主要是因为计算提取函数的时候稍微耗时. 注意到条件函数依赖的时间开销比较低，主要是因为唯一的条件函数依赖 φ_3 只匹配到很少的数据.

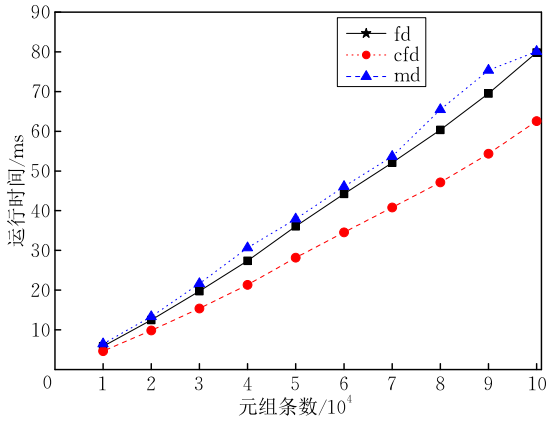

图 4 检测一条约束平均时间随元组数的变化，错误率＝5%

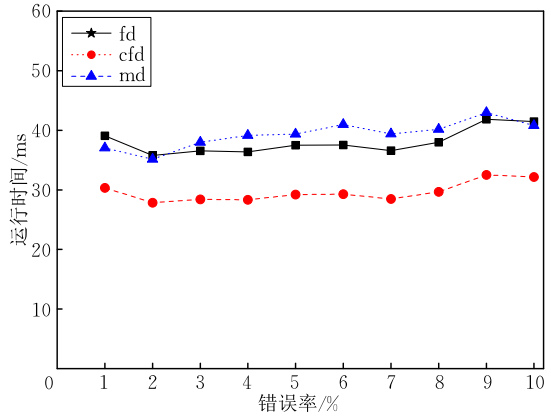

图 5 检测一条约束平均时间随出错概率的变化，元组条数＝5×10^4

图 6 描述了每发现一个错误数据，所花的平均时间，从图中可以看出，微函数依赖在单个错误上的平均时间开销和条件函数依赖接近，远低于函数依赖. 即微函数依赖均摊到每个错误上的时间成本是远远低于函数依赖和条件函数依赖的.

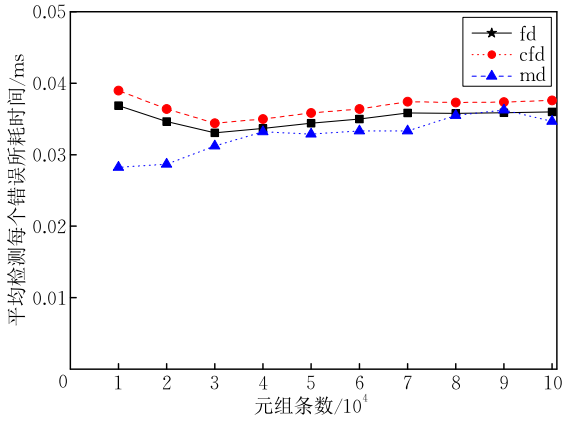

图 6 发现一个错误的平均运行时间随元组数的变化，错误率＝5%

在共享排序的实验中，约束 φ_0 和 φ_4 以及 φ_2 和 φ_8 之间是可以进行排序共享的. 图 7 和图 8 显示了在微函数依赖的冲突检测中，共享排序的优化策略对时间开销的影响，可以看出优化效果还是十分明显的，大约节省 20%的时间.

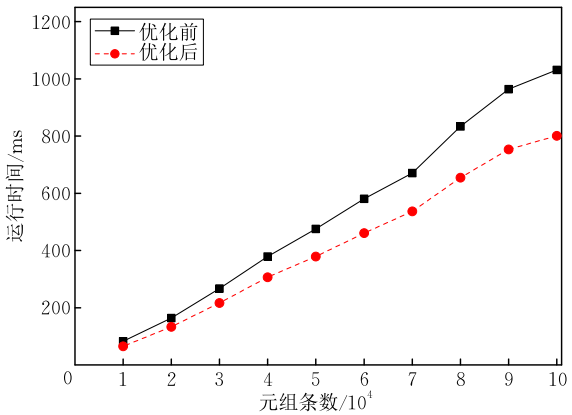

图 7　优化技术对运行速度的影响：检测一条约束平均时间随元组数的变化，错误率＝5%

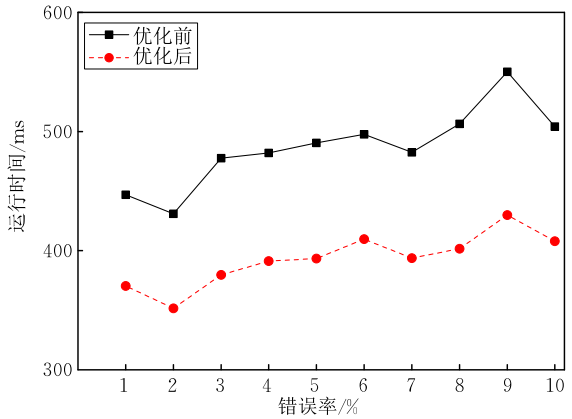

图 8　优化技术对运行速度的影响：检测一条约束平均时间随出错概率的变化，元组条数＝50 000

从上述实验结果可以看到，在可接受的时间代价内，微函数依赖能够检测出比以前方法多得多的错误数据. 而且本文提出的基于排序的方法在经过排序操作共享的优化之后，时间效率有很明显的提高.

7　相关工作

为了解决数据的不一致问题，已有研究主要有两个方向，第一个方向是找出一个满足依赖的最小修复[15]，此处“最小”的含义是，删除或插入最少的元组，或者更改最少的属性值，或者产生最小的修复代价. 这些方法都被证明是难解问题. 另一类方法是一致查询[16]，其含义是：给定一个查询，找出该查询在所有极小修复上查询结果的交集. 很显然，这种方法比前者的复杂性更高.

为了解决传统函数依赖和条件函数依赖修复数据的难点，文献[10,17-18]主要研究了基于编辑规则和主数据的编辑距离的数据修复方法，这种规则假定主数据是完整而且正确的，如果一条元组与主数据中的某条元组在某些特定属性上的值满足一定的距离限制，那么它在另外某属性上也应该和主数据相同. 基于主数据和编辑距离的方法不仅能够方便快速地发现数据错误，还能够直观地指导数据修复，避免了(条件)函数依赖方法中的修复难解问题. 然而，这种方法的假定场景比较强，它需要存在一个完全可靠的高质量主数据，而在实际应用中，这样的条件是未必能够满足的. 此外，同一研究组又提出了能够得到可靠修复的修复规则[11]，并对其可满足性做了分析. 修复规则在包含约束语义的同时，还包含了如何进行修复的信息，比如，关系模式(国家，首都)上的规则(“中国”，{“上海”，“香港”}，“北京”)表示的语义是，如果国家名称为“中国”，首都名称为“上海”或者“香港”，那么这条记录是错误的，因为可以看出该记录是关于中国的，那么首都应该被修改成“北京”. 修复依赖同基于主数据的编辑规则类似，也是把大量的可靠信息引入规则中，从而保障能够得到确定可靠的修复结果.

除此之外，还有更多的约束类型被提出来：文献[13]提出了“差分依赖”，它描述：如果两个元组在某一个属性上值的距离足够近，那么在另外一个属性上，它们的值也足够接近，例如：约束$[Date(\leq 7)]\rightarrow[Price(<100)]$表示 7 天之内物品的价格波动不能够超过 100 元. 文章首先解决了几个理论问题：差分函数之间的包含关系、蕴含问题、闭包、正确且完备的推理系统、极小覆盖等. 并证明了这类依赖的挖掘问题属于难解问题，给出启发式搜索算法以及有效的剪枝策略. 为了研究数据空间中异构数据之间的数据依赖，文献[14]定义了一个一般化的约束形式：可比较约束. 它制定了可比属性之间的约束. 它包含了一个数据库领域的一个广泛约束类. 包括函数依赖，度量函数依赖等. 文章研究了约束的可满足性，证明它是难解问题，并给出贪心和随机算法计算最大可满足性子集. 文献[19]针对异构数据源中由数据格式不一致引发的一致性错误，利用描述属性值相似性测度扩充了函数依赖，用来描述异构数据的一致性约束，发现和修复异构数据的一致性错误. 文献[20]提出了空间语义完整性约束形式化定义，能

够对现实中的约束进行统一描述,是对传统函数依赖和包含依赖的扩展,更重要的是考虑了空间属性,对空间特征之间加上了拓扑关系.文章研究了该种约束的可满足性,证明该问题属于难解问题,也给出了非难解情况下的算法,来检查约束集的可满足性,对于难解情况,给出近似算法存在的条件.文献[21]在有时间戳的数据上提出了序列依赖语义规则,用来描述随时间变化数据的一致性约束,试图解决随时间变化数据的一致性错误的发现和修复问题.

此外,对于实体也有一类一致性描述方法,即实体同一性或实体识别(又称记录匹配,元组匹配),它是信息质量方面研究最多的领域.针对存储在关系数据库中的信息,人们已提出了多个实体识别算法[22-30].这些工作大多偏重于如何提高信息中识别实体的效率,如分块技术[31]和滑动窗口技术[32].市场上的信息清洗商业系统也大多支持实体识别的功能.然而,实体识别方法仍然存在如下问题:(1)需要相关领域专家的人工参与,或依赖于概率式启发式规则或学习式的启发式规则,工作量大,自动化程度低;(2)现有的实体识别算法大多缺乏理论基础,很难与解决信息质量其他问题的技术融合;(3)现有实体识别技术多专注于存储在关系数据库中的信息,鲜有针对复杂结构信息的实体识别技术.

为解决前两个问题,Fan 等人[33-34]也提出了一个匹配约束理论,包括描述匹配规则的约束语言、推理机制、公理系统和匹配键的推导算法,有效地自动确定匹配键,并使信息匹配操作能够与基于约束的其他信息质量问题的处理方法有效地结合.人们已开发了自动发现匹配约束的算法[35].

8 结论及未来工作

文章将已有的函数依赖和条件函数依赖进行了扩展,提出更一般化的微函数依赖.静态分析方面,证明了微函数依赖的可满足问题是 NP-完全的,蕴含问题是 CoNP-完全的.另外,给出了由五条推理规则构成的推理系统,并证明了该系统的正确性以及完备性.最后,文章通过实验验证了微函数依赖能够检测出更多的错误数据,而时间开销的增加在可接受的范围内.

本文主要从理论上讨论了微函数依赖的性质.除此之外,还有如下几个方面需要进一步研究:(1)微函数依赖的挖掘问题,为了充分发挥其在数据修复中的作用,有必要给出足够的微函数依赖.仅仅靠用户手动给出是不现实的,应该针对不同类别的函数给出不同的挖掘方法;(2)本文假设固有函数依赖都由用户给出,假设条件略强,可以针对不同类别的函数,分别研究其固有函数依赖集是否能够自动生成,如果可以,是否存在有效算法等;(3)在本文基于排序的算法中,约束之间的排序共享是直接人工给出的,当约束个数很多时,不同的共享策略之下是否有不同的排序次数.如果排序次数受共享策略的影响,则需要研究如何找出最优的排序共享策略,使得排序次数最少,检测错误所花的时间最低;(4)提出微函数依赖的最终目的,是为了进行数据修复,本文实验中只给出了最直观的方法,面临日益严峻的大数据问题,如何快速有效地利用微函数依赖发现并修复数据库中的不一致,也是值得进一步研究的问题.

参 考 文 献

[1] Redman T C. The impact of poor data quality on the typical enterprise. Communications of the ACM, 1998, 41(2): 79-82

[2] Swartz N. Gartner warns firms of 'dirty data'. Information Management Journal, 2007, 41(3): 6

[3] Otto B. From Health Checks to the Seven Sisters: The Data Quality Journey at BT [Ph. D. dissertation]. University of St. Gallen, St. Gallen, Switzerland, 2009

[4] Eckerson W W. Data quality and the bottom line. Washington: The Data Warehouse Institute, Technical Report, 2002

[5] Shilakes C C, Tylman J. Enterprise information portals. The Electronic Library, 2000, 18(5): 354-362

[6] Codd E F. A relational model of data for large shared data banks. Communications of the ACM, 1970, 13(6): 377-387

[7] Fan W, Geerts F, Jia X, et al. Conditional functional dependencies for capturing data inconsistencies. ACM Transactions on Database Systems, 2008, 33(2): 1-44

[8] Bohannon P, Fan W, Geerts F, et al. Conditional functional dependencies for data cleaning//Proceedings of the 23rd IEEE International Conference on Data Engineering. Istanbul, Turkey, 2007: 746-755

[9] Bravo L, Fan W, Geerts F, et al. Increasing the expressivity of conditional functional dependencies without extra complexity //Proceedings of the 24th IEEE International Conference on Data Engineering. Cancun, Mexico, 2008: 516-525

[10] Fan W, Li J, Ma S, et al. Towards certain fixes with editing rules and master data//Proceedings of the 36th Very Large Data Bases Conferencee. Singapore, 2010: 173-184

[11] Wang J, Tang N. Towards dependable data repairing with fixing rules//Proceedings of the ACM International Conference on Management of Data. Snowbird, USA, 2014: 457-468

[12] Chen W, Fan W, Ma S. Analyses and validation of conditional dependencies with built-in predicates//Proceedings of the 20th International Workshop on Database and Expert Systems Applications. Berlin, Germany, 2009: 576-591

[13] Song S, Chen L. Differential dependencies: Reasoning and discovery. ACM Transactions on Database Systems, 2011, 36(3): 1-41

[14] Song S, Chen L, Yu P S. Comparable dependencies over heterogeneous data. The International Journal on Very Large Data Bases, 2013, 22(2): 253-274

[15] Baudinet M, Chomicki J, Wolper P. Constraint-generating dependencies//Proceedings of the 5th International Conference on Database Theory. Prague, Czech Republic, 1995: 322-337

[16] Arenas M, Bertossi L, Chomichi J. Consistent query answers in inconsistent databases//Proceeding of the 8th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. New York, USA, 1999: 68-79

[17] Yu W. Improving Data Quality: Data Consistency, Deduplication, Currency and Accuracy [Ph. D. dissertation]. University of Edinburgh, Edinburgh, Scotland, 2013

[18] Fan W F, et al. CerFix: A system for cleaning data with certain fixes. Proceedings of the Very Large Data Bases Endowment, 2011, 4(12): 1375-1378

[19] Koudas N, et al. Metric functional dependencies//Proceedings of the 25th International Conference on Data Engineering. Shanghai, China, 2009: 1275-1278

[20] Bravo L, Rodriguez M A. Formalization and reasoning about spatial semantic integrity constraints. Data & Knowledge Engineering, 2012, 72(1): 63-82

[21] Golab L, et al. Sequential dependencies. Proceedings of the Very Large Data Bases Endowment, 2009, 2(1): 574-585

[22] Yancey W E. Bigmatch: A program for extracting probable matches from a large file for record linkage. Statistical Research Division US Bureau of the Census, Washington DC: Technical Report RRC2002/01, 2002

[23] Shen W, Li X, Doan A. Constraint-based entity matching//Proceedings of the 20th National Conference on Artificial Intelligence. Pittsburgh, USA, 2005: 862-867

[24] Whang S E, Benjelloun O, Garcia M H. Generic entity resolution with negative rules. The International Journal on Very Large Data Bases, 2009, 18(6): 1261-1277

[25] Weis M, et al. Industry-scale duplicate detection. Proceedings of the Very Large Data Bases Endowment, 2008, 1(2): 1253-1264

[26] Arasu A, Ré C, Suciu D. Large-scale deduplication with constraints using dedupalog//Proceedings of the 25th International Conference on Data Engineering. Shanghai, China, 2009: 952-963

[27] Hernández M A, Stolfo S J. Real-world data is dirty: Data cleansing and the merge/purge problem. Data Mining and Knowledge Discovery, 1998, 2(1): 9-37

[28] Chaudhuri S, et al. Robust and efficient fuzzy match for online data cleaning//Proceedings of the ACM International Conference on Management of Data. San Diego, USA, 2003: 313-324

[29] Benjelloun O, et al. Swoosh: A generic approach to entity resolution. The International Journal on Very Large Data Bases, 2009, 18(1): 255-276

[30] Arasu A, Chaudhuri S, Kaushik R. Transformation-based framework for record matching//Proceedings of the 24th International Conference on Data Engineering. Cancún, México, 2008: 40-49

[31] Herzog T N, Scheuren F J, Winkler W E. Data Quality and Record Linkage Techniques. New York, USA: Springer Publishing Company, Inc., 2007

[32] Elmagarmid A K, Ipeirotis P G, Verykios V S. Duplicate record detection: A survey. IEEE Transactions on Knowledge and Data Engineering, 2007, 19(1): 1-16

[33] Fan W, et al. Dynamic constraints for record matching. The International Journal on Very Large Data Bases, 2011, 20(4): 495-520

[34] Fan W, et al. Reasoning about record matching rules. Proceedings of the Very Large Data Bases Endowment, 2009, 2(1): 407-418

[35] Song S, Chen L. Discovering matching dependencies//Proceedings of the 18th ACM Conference on Information and Knowledge Management. Hong Kong, China, 2009: 1421-1424

SUN Ji-Zhou, born in 1985, Ph. D. candidate. His research interests are data quality management and approximate computing over weak available massive data.

LI Jian-Zhong, born in 1950, Ph. D., professor, Ph. D. supervisor. His research interests include database, massive data processing, wireless sensor networks, IoT, etc.

GAO Hong, born in 1966, Ph. D., professor, Ph. D. supervisor. Her research interests include wireless sensor networks, IoT, massive data management and data mining, etc.

LIU Xian-Min, born in 1984, Ph. D., lecturer. His research interests include massive data computing and data quality management.

Background

Dirty data is becoming more inevitable and widespread, which often causes serious consequences and is often expensive to clean. In recent years, the database community has investigated extensively the problem of dealing with dirty data. Inconsistency is one of the most important aspects of dirty data. A database is inconsistent if it violates some data quality rules.

The severe condition of inconsistent data requires comprehensive classes of quality rules to help detecting and repairing dirty data. To this end, several kinds of rules such as functional dependencies, conditional functional dependencies, extended conditional functional dependencies, conditional functional dependencies with built-in predicates, fixing rules, editing rules, differential dependencies and comparable dependencies etc were used to detect and repair inconsistent data. Several of them have been studied deeply in terms of satisfiability, implication and inference systems. However, all the proposals treat each attribute entirely, even though there's much inner information in an attribute.

To this end, this paper studied a new kind of dependencies, called micro dependencies(MDs), which can catch the inner-attribute relationships in a database. By doing this, much more data errors can be detected with acceptable time consuming. This paper mainly studied the static properties of MDs, including satisfiability analysis and implication analysis. For inference system, five inference rules were given, and the completeness and soundness of this inference system can be ensured.

This research is supported by the National Basic Research Program (973 Program) of China under Grant No. 2012CB316202.

Our group has been focusing on the research of data management for a long period. Many outstanding works have been published in worldwide conferences, journals and transactions, including SIGMOD, VLDB, ICDE, KDD, INFOCOM, TKDE, VLDB Journal et al. This paper proposed a new kind of data quality rules, which can help clean dirty data more thoroughly.