不完全信息环境下XML Schema规范化研究

殷丽凤[1] 郝忠孝[1,2]
(哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)[1]
(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)[2]

摘 要 为了解决不完全信息环境下XML模式设计中XML文档的数据冗余和操作异常,研究了不完全信息环境下XML Schema的规范化问题,提出了XML Schema和符合XML Schema的不完全XML文档树等概念;基于节点等价、节点相容、节点信息等价和节点信息相容等概念提出了XML强函数依赖的定义,给出了相应的推理规则;给出了求路径集强闭包和成员籍问题的算法,并对算法的正确性进行了证明,对其时间复杂度进行了分析。提出了不完全信息环境下XML范式和相应的规范化算法。研究成果较好地解决了数据冗余问题,避免了更新异常现象,更好地实现了XML Schema设计。

关键词 不完全信息,XML强函数依赖,路径集强闭包,不完全信息环境下的XML范式
中图法分类号 TP311.13 **文献标识码** A

Research of Normalization for XML Schema under Incomplete Information Circumstances

YIN Li-feng[1] HAO Zhong-xiao[1,2]
(College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China)[1]
(College of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China)[2]

Abstract For solving data redundancies and abnormal manipulation for XML documents in schema design under incomplete information circumstances, the normalization theory of XML Schema under incomplete information circumstances was discussed. The concepts of XML Schema and incomplete XML document tree according with XML Schema were formalized. Based on the equivalence of the nodes, the consistency of the nodes, the equivalence of the nodes' information and the consistency of the nodes' information, XML strong functional dependency's definition was given, inference rules for XML strong functional dependency were presented. The arithmetic of path set strong closure and membership problem was proposed, its correctness was proved and its time complication was analyzed. The definition of XML normal form under incomplete information circumstances and the corresponding arithmetic of the normalization were formalized. The production in this work removes the redundancies of datum, eliminates update anomalies and achieves better design of XML Schema.

Keywords Incomplete information, XML strong functional dependency, Path set strong closure, XML normal form under incomplete information circumstances

1 引言

XML已经成为Internet上数据表示和交换的标准,但XML许多技术正处于研究阶段,其中XML的模式设计是一个很重要的问题。一个好的模式设计能够减少数据冗余以及避免更新、插入和删除异常,从而减少存储空间。目前DTD和XML Schema是XML最主要的模式,有许多作者已经对基于DTD的XML模式设计进行了研究。在国内,文献[1-3]在完全信息环境下提出了基于DTD的范式和相应的规范化算法,但还没有涉及到不完全信息;在国外,M. Arenas和L. Libkin[4]基于树元组定义了XML函数依赖,给出了XML范式和基于DTD的规范化算法。以上理论研究为基于DTD的XML模式设计奠定了基础。M. W. Vincent和Liu Jixue等人[5-7]基于XML文档提出了XML多值依赖定义和相应的范式理论,在不完全信息环境下定义了XML强函数依赖并且研究了相应的范式理论,但没有说明不完全信息的具体语义。

DTD存在一些缺陷,如不支持数据类型、扩展性较差等。W3C针对DTD的缺陷设计了XML另一种模式XML Schema,2001年5月W3C正式推荐XML Schema成为XML模式的标准。XML Schema符合XML规范,可以直接用XML的API(如DOM,SAX)进行解析,不但支持命名空间,具有丰富的数据类型和属性组等,而且支持不完全信息。由于客观世界存在大量不完全信息,XML Schema能够更好地描述现实客观世界,所以对XML Schema的设计研究更具有现实应

到稿日期:2008-11-11 返修日期:2009-02-09 本文受黑龙江省自然科学基金(F200702)资助。
殷丽凤(1976—),女,博士研究生,主要研究领域为不完全信息环境下XML数据库理论和主动XML数据库理论,E-mail:yinlife0598@sina.com;郝忠孝(1940—),男,教授,博士生导师,主要研究领域为数据库系统与理论。

用意义。但 XML Schema 只可以表示不完全信息，却没有说明不完全信息的具体语义。为了说明不完全信息的语义，本文将文献[8]中的存在型不完全信息的语义(即存在型不完全信息必然对应某个或几个实值)引入 XML Schema 中，由于存在型不完全信息的引入，完全信息环境下的 XML 函数依赖、键、多值依赖的定义都失去了原来的意义，故不完全信息环境下的 XML 模式设计理论不能用完全信息环境下的相应理论解决问题。

作者在不完全信息环境下做了一些工作，对 XML 强函数依赖(记作 XSFD)[9]和 XML 强闭包依赖进行了研究[10]。本文基于 XML Schema、符合 XML Schema 的不完全 XML 文档树、节点信息等价、节点信息相容等概念提出了 XML 强函数依赖的定义和相应的推理规则集，给出了不完全信息环境下路径集强闭包和成员籍问题的算法。最后提出了不完全信息环境下的 XML 范式的定义和转换 XML Schema 为不完全信息环境下 XML 范式的规范化算法，实现了更好的 XML Schema 设计。

2 基本概念

为了形式化地描述 XML 强函数依赖，首先给出基本概念，其中 ϕ 为存在型不完全信息，简记为不完全信息。由于 XML Schema 的复杂类型元素是对 XML Schema 的结构进行约束，简单类型元素和属性才能体现出数据信息，所以本文限制不完全信息体现在简单类型元素和属性中。另外，为了方便研究问题，对复杂类型元素的子元素限定为不包含混合类型。

定义 1(XML Schema) XML Schema 为一个九元组 $S=(CE,SE,A,D,CP,SP,R,Type,r)$。其中，(1)$CE$ 表示复杂类型元素的有限集合。(2)SE 为简单类型元素的有限集合。(3)A 为属性的有限集合。(4)D 为数据的标识。(5)CP 表示从复杂类型元素到子元素的映射，$\forall\tau\in CE, CP(\tau)$表示正则表达式 $\alpha=\varepsilon|\tau'|\alpha,\alpha|\alpha^*$，其中 ε 表示空的序列，$\tau'\in CE\cup SE$，“,”表示连接，“$*$”表示克林闭包，$\alpha\subseteq CE\cup SE$。(6)$SP$ 表示从简单类型元素到 D 的映射。(7)R 表示从复杂类型元素到属性的映射，$\forall\tau\in CE, R(\tau)$表示正则表达式 $\alpha=\varepsilon|@a|\beta,\beta$，其中 ε 表示空的序列，$@a\in A$，“,”表示连接，$\beta\subseteq A$。(8)$Type$ 表示 $CE\cup SE\cup A\cup D$ 所对应的数据类型和数据约束。(9)r 表示 Schema 根元素，$r\in CE$，$\forall\tau\in CE$，r 不会出现在 $CP(\tau)$ 中。

定义 2(XML Schema 路径) 给定一个 XML Schema S，字符串 $p=w_1/w_2/\cdots/w_n$，若 $w_1=r, CP(w_i)\in CE$，其中 $i\in[1,n-2], w_n\in R(W_{n-1})$，称 p 为 S 中的属性类型路径；若 $w_1=r, CP(w_i)\in CE\cup SE$，其中 $i\in[1,n-2], w_n=D$，称 p 为 S 中的值类型路径；若 $w_1=r, CP(w_i)\in CE$，其中 $i\in[1,n-2]$，$w_n\in CE$，称 p 为 S 中的复杂元素类型路径；若 $w_1=r, CP(w_i)\in CE$，其中 $i\in[1,n-2], w_n\in SE$，称 p 为 S 中的简单元素类型路径。$Paths(S)$表示 S 上所有路径的集合，$Dpaths(S)$表示 S 上值类型路径集合，$Apaths(S)$表示 S 上属性类型路径集合，$CEpaths(S)$表示 S 上复杂元素类型路径集合，$SEpaths(S)$表示 S 上简单元素类型路径集合，$Epaths=CEpaths(S)\cup SEpaths(S)$表示 S 上元素类型路径集合。$Parent(w_i)$表示路径 $w_1/w_2/\cdots/w_i$ 的父路径 $w_1/w_2/\cdots/w_{i-1}$ 的最后标识，即 $Parent(w_i)=w_{i-1}$。

定义 3(符合 XML Schema 的不完全 XML 文档树) 给定 XML Schema 为 $S=(CE,SE,A,D,CP,SP,R,Type,r)$，称七元组 $T=(V,lab,cele,sele,att,val,vr)$为符合 S 的不完全 XML 文档树。其中，(1)V 表示节点的集合。(2)lab 表示从 V 到 $CE\cup SE\cup A\cup D$ 的映射，若 $lab(v)\in CE$，称 v 为复杂元素类型节点；若 $lab(v)\in SE$，称 v 为简单元素类型节点；若 $lab(v)\in A$，称 v 为属性类型节点，若 $lab(v)=D$，称 v 为值类型节点。(3)$cele$ 表示从复杂元素类型节点到复杂元素类型节点或简单元素类型节点的映射，若 $v\in CE, cele(v)=[v_1,\cdots,v_n]$，满足 $lab(v_i)\in CP(lab(v))$，其中 $i\in[1,n]$。(4)$sele$ 表示从简单类型元素节点到值类型节点的映射；若 $v\in SE$，$sele(v)=v'$，满足 $lab(v')=D$。(5)att 表示从复杂元素类型节点到属性类型节点的映射，若 $v\in CE, att(v)=[v_1,\cdots,v_m]$，满足 $lab(v_i)\in R(lab(v))$，其中 $i\in[1,m]$。(6)若 v 为值类型节点和属性类型节点，val 返回相应类型的数据且满足相应的数据约束，若数据约束允许不完全信息，则相应的数据可能为 ϕ，即 $lab(v)\in A$ 或 $lab(v)=D$，$val(v)$返回 v 的值且满足相应的 $Type(lab(v))$；若 $v\in CE\cup SE$，则 $val(v)=v$。(7)v_r 为文档树的根节点，且 $lab(v_r)=r$。

定义 4(路径节点集) 设 T 为符合 S 的不完全 XML 文档树，$p\in Paths(S)$，$p=w_1/w_2/\cdots/w_n$，$n=v_1,v_2,\cdots,v_n$ 为 T 中的节点序列。若 $lab(v_1)=w_1,\cdots,lab(v_n)=w_n$，称 n 为通过路径 p 的路径节点集，$[[p]]$表示 T 中通过路径 p 的路径节点集的最后节点的集合。

定义 5(T 的一对一拆分) 设 T 为符合 S 的不完全 XML 文档树，$Paths(S)$为 S 上所有路径的集合，$T'\subseteq T$，若 $\forall p\in Paths(S)$，则 $\exists s\in T'$，s 为通过路径 p 的路径节点集且 s 唯一；反之，若 $\forall s\in T'$，则 $\exists p\in Paths(S)$，s 为通过路径 p 的路径节点集且 p 唯一，称 T' 与 $Paths(S)$为一对一关系，称每个与 $Paths(S)$为一对一关系的 T'为 T 关于 $Paths(S)$的一个一对一拆分，所有一对一拆分的并为 T。本文所提到的 T 就是由 T 关于 $Paths(S)$的所有一对一拆分所构成的，记作 $\{T\backslash Paths(S)\}$。

定义 6(节点信息等价) 设 T 为符合 S 的不完全 XML 文档树，$p\in Dpaths(S)\cup Apaths(S)$，$n_1,n_2\in[[p]]$，若满足下面的条件之一，则称节点信息等价，记作 $n_1\doteqdot_{in}n_2$；否则称节点信息不等价，记作 $n_1\not\doteqdot_{in}n_2$。

1)$val(n_1)$和 $val(n_2)$均为完全信息，则 $val(n_1)=val(n_2)$；

2) $val(n_1)$和 $val(n_2)$都为不完全信息，它们的语义信息相同。进行不完全信息代换时，$val(n_1)$和 $val(n_2)$代换为同一完全信息；

定义 7(节点信息相容) 设 T,S,p,n_1,n_2 的意义同定义 6。若 n_1,n_2 满足下面的条件之一，则称节点信息相容，记作 $n_1\doteq_{in}n_2$；否则称节点信息不相容，记作 $n_1\not\doteq_{in}n_2$。

1) $n_1\doteqdot_{in}n_2$；

2)若 $val(n_1)$为不完全信息，$val(n_2)$为完全信息，至少存在一个不完全信息代换过程，使 $val(n_1)=val(n_2)$成立；

3)若 $val(n_1)$和 $val(n_2)$都为不完全信息，且它们限定的代换范围的交集不是空的，即至少有一种不完全信息代换过程，使 $val(n_1)=val(n_2)$成立；

定义 8(节点等价/节点相容) 设 T 为符合 S 的不完全 XML 文档树，$p\in Epaths(S)$，$n_1,n_2\in[[p]]$，若 $val(n_1)=val(n_2)$，则称节点相等(节点重合)，相等的节点也称节点等价和节点相容，记作 $n_1\doteqdot_v n_2$ 和 $n_1\doteq_v n_2$；若 $val(n_1)\neq val(n_2)$，则称节点不相等(节点不重合)，不相等的节点也称节点不等价和节点不相容，记作 $n_1\neq_v n_2$ 和 $n_1\not\doteq_v n_2$。

定义 9(XML 强函数依赖) 给定 XML Schema S，在 S 上的 XML 强函数依赖(记作 XSFD)ψ 具体表现形式为 $x_1,\cdots,x_k\xrightarrow{S}y_1,\cdots,y_m$，其中，$x_1,\cdots,x_k$ 为 ψ 的左部路径集，$x_1,\cdots,x_k\in Paths(S)$，$y_1,\cdots,y_m$ 为 ψ 的右部路径集，$y_1,\cdots,y_m\in Paths(S)$。符合 S 的不完全 XML 文档树为 T，对{$T\backslash Paths(S)$}中的任意两个子树 T_1,T_2(T_1 和 T_2 可以相同)，满足下面的条件之一，则称 T 满足 XML 强函数依赖 ψ。

(1)当 $x_1,\cdots,x_k,y_1,\cdots,y_m\in Dpaths(S)\cup Apaths(S)$时，若 $v_{1i}\doteqdot_m v_{2i}$，其中 $v_{1i}\in[[x_i]]$且 $v_{1i}\in T_1$，$v_{2i}\in[[x_i]]$且 $v_{2i}\in T_2$，$i\in[1,k]$，那么 $v_{1j}\doteqdot_m v_{2j}$，其中 $v_{1j}\in[[y_j]]$且 $v_{1j}\in T_1$，$v_{2j}\in[[y_j]]$且 $v_{2j}\in T_2$，$j\in[1,m]$；

(2)当 $x_1,\cdots,x_k\in Dpaths(S)\cup Apaths(S)$，$y_1,\cdots,y_m\in Epaths(S)$时，若 $v_{1i}\doteqdot_m v_{2i}$，其中 $v_{1i}\in[[x_i]]$且 $v_{1i}\in T_1$，$v_{2i}\in[[x_i]]$且 $v_{2i}\in T_2$，$i\in[1,k]$，那么 $v_{1j}\doteq_v v_{2j}$，其中 $v_{1j}\in[[y_j]]$且 $v_{1j}\in T_1$，$v_{2j}\in[[y_j]]$且 $v_{2j}\in T_2$，$j\in[1,m]$；

(3)当 $x_1,\cdots,x_k\in Epaths(S)$，$y_1,\cdots,y_m\in Dpaths(S)\cup Apaths(S)$时，若 $v_{1i}\doteq_v v_{2i}$，其中 $v_{1i}\in[[x_i]]$且 $v_{1i}\in T_1$，$v_{2i}\in[[x_i]]$且 $v_{2i}\in T_2$，$i\in[1,k]$，那么 $v_{1j}\doteqdot_m v_{2j}$，其中 $v_{1j}\in[[y_j]]$且 $v_{1j}\in T_1$，$v_{2j}\in[[y_j]]$且 $v_{2j}\in T_2$，$j\in[1,m]$；

(4)当 $x_1,\cdots,x_s(s<k),y_1,\cdots,y_t(t<m)\in Dpaths(S)\cup Apaths(S)$，$x_{s+1},\cdots,x_k,y_{t+1},\cdots,y_m\in Epaths(S)$时，若 $v_{1i}\doteq_m v_{2i}$且 $v_{1a}\doteq_v v_{2a}$，其中 $v_{1i}\in[[x_i]]$且 $v_{1i}\in T_1$，$v_{2i}\in[[x_i]]$且 $v_{2i}\in T_2$，$i\in[1,s]$，$v_{1a}\in[[x_a]]$且 $v_{1a}\in T_1$，$v_{2a}\in[[x_a]]$且 $v_{2a}\in T_2$，$a\in[s+1,k]$，那么 $v_{1j}\doteqdot_m v_{2j}$ 且 $v_{1b}\doteq_v v_{2b}$，其中 $v_{1j}\in[[y_j]]$且 $v_{1j}\in T_1$，$v_{2j}\in[[y_j]]$且 $v_{2j}\in T_2$，$j\in[1,t]$，$v_{1b}\in[[x_b]]$且 $v_{1b}\in T_1$，$v_{2b}\in[[x_b]]$且 $v_{2b}\in T_2$，$b\in[t+1,m]$。

3 不完全信息环境下的成员籍问题

基于上节提出的概念，本节首先给出 XML 强函数依赖的推理规则，然后提出不完全信息环境下求路径集强闭包和成员籍问题的算法。

定理 1 给定 XML Schema $S=(CE,SE,A,D,CP,SP,R,Type,r)$，$S$ 上的路径集合为 $Paths(S)$。若 $X,Y,Z,W\subseteq Paths(S)$，下述推理规则成立：

(1)分解规则。若 $X\xrightarrow{S}Y$ 在 S 上成立，$W\subseteq Y$，则 $X\xrightarrow{S}W$ 在 S 上成立。

(2)合并规则。若 $X\xrightarrow{S}Y$ 在 S 上成立，$X\xrightarrow{S}Z$ 在 S 上成立，则 $X\xrightarrow{S}YZ$ 在 S 上成立。

(3)传递规则。若 $X\xrightarrow{S}Y$ 在 S 上成立，$Y\xrightarrow{S}Z$ 在 S 上成立，则 $X\xrightarrow{S}Z$ 在 S 上成立。

(4)伪传递规则。若 $X\xrightarrow{S}Y$ 在 S 上成立，$WY\xrightarrow{S}Z$ 在 S 上成立，则 $WX\xrightarrow{S}Z$ 在 S 上成立。

(5)属性不重名规则。若 $X\xrightarrow{S}y$ 成立，$y\in CEpaths(S)$ 且存在 $y.@a$，则 $X\xrightarrow{S}y.@a$ 成立。

(6)若 $X\xrightarrow{S}y$ 成立，$y\in SEpaths(S)$，则 $X\xrightarrow{S}y.D$ 成立。

(7)唯一的根节点。对 $\forall p\in Paths(S)$，只要 p 不为空，则 $p\xrightarrow{S}r$ 成立。

定义 10(逻辑蕴涵) 设 XML Schema $S=(CE,SE,A,D,CP,SP,R,Type,r)$，$Paths(S)$为 S 上的路径集合，F 为 S 上的 XSFD 的集合，若对任意符合 S 的不完全 XML 文档树都满足 XSFD：$X\xrightarrow{S}Y$，则称 F 逻辑蕴涵 $X\xrightarrow{S}Y$，或称 $X\xrightarrow{S}Y$ 可由 F 推出。

定义 11(路径集的强闭包) 设 XML Schema $S=(CE,SE,A,D,CP,SP,R,Type,r)$，$Paths(S)$为 S 上的路径集合，F 为 S 上的 XSFD 的集合，$X\subseteq Paths(S)$，则 X 关于 XSFD 集合 F 的强闭包 $X_s^+=\{Z\mid X\xrightarrow{S}Z$ 可由推理规则(1)－(7)推出}。

为了求解不完全信息环境下的成员籍问题，下面根据推理规则给出求关于 XSFD 集的路径集强闭包的算法。

算法 1 Pathset-strong-closure{求路径集 X 的强闭包算法}。

```
输入:路径集 X={x1,…,xn},S 上的 XSFD 的集合 F
输出:Xs+
Pathset-strong-closure(X,F)
begin
(1)Xs+:=Φ; //初始化为空;
(2)for 每个 V→S Y∈F do
    if V=X then
        Xs+:= Xs+∪Y ;
    if V⊂X then
        for 每个 P→S Q∈F do
            if W=X−V 且 P=W∪Y then
                Xs+:= Xs+∪Q;
(3)X(0):= Φ; //初始化为空;
    X(1):= Xs+;
(4) while X(0)≠X(1)
    X(0):=X(1);
    for 每个 P→S Q∈F do
        if P⊆X(1) then
            X(1):= X(1)∪Q;
(5) Xs+:= X(1);
(6)for Xs+ 中每一个元素类型路径 p do
    if p.@a∈Paths(S) then
        Xs+:=Xs+∪p.@a;
    if p.D∈Paths(S) then
        Xs+:= Xs+∪p.D;
(7) Xs+:=Xs+∪r;
(8)return(Xs+);
end.
```

定理 2 算法 1 是正确的，其时间复杂度为 $O(n^2+m)$，n 为 F 中 XSFD 的个数，m 为 $Paths(S)$中元素类型路径的个数。

证明：正确性。算法的第(1)步初始化 X_s^+ 为空；第(2)步把左部路径集为 X 的 XSFD 的右部路径集加入 X_s^+ 中，把满

足伪传递规则 XSFD 的右部路径集加入 X_s^+ 中；第(4)、(5)步实现满足传递规则的 XSFD 右部路径集加入 X_s^+ 中。第(6)步根据推理规则(5)和(6)对所求得的 X_s^+ 中的复杂元素类型路径 p，把相应 $p.@a$ 加入 X_s^+ 中，对所求得的 X_s^+ 中的简单元素类型路径 p，把相应 $p.D$ 加入 X_s^+ 中；第(7)步把根路径 r 加入 X_s^+ 中，所以通过此算法所求得的路径集 X 的强闭包都是由推理规则集(1)—(7)所推出的，满足定义，所以是正确的。

时间复杂度的分析。算法的时间复杂度主要由第(2)步、第(4)步和第(6)步的循环次数决定。第(2)步循环的次数由双层循环决定，内、外层循环执行的次数由 F 中 XSFD 的个数决定，即为 n，最坏情况下总共执行次数为 n^2；算法第(4)步每一轮扫描所需要的时间为 n，最坏情况是每轮扫描只有一个 XSFD 满足条件，只有一个新的路径被加入 $X(1)$，共循环 n 轮，于是第(4)步所用的时间总共为 n^2；第(6)步循环所用的时间最多为 $Paths(S)$ 中元素类型路径的个数，即为 m，算法总共执行的次数为 $2n^2+m$，所以算法的时间复杂度为 $O(n^2+m)$。

给定一个 XSFD $X\xrightarrow{S}Y$，F 能否推出 XSFD $X\xrightarrow{S}Y$，此问题为不完全信息环境下的成员籍问题，下面给出不完全信息环境下成员籍问题的算法。

算法 2 Membership{求成员籍问题算法}

输入：$X\xrightarrow{S}Y,F$

输出：true 或 false

```
Membership(X -S-> Y, F)
begin
  X_s^+ := Pathset-strong-closure(X, F);
  if Y∈X_s^+ then
    return(true);
  else
    return(false);
end.
```

定理 3 算法 2 是正确的，其时间复杂度为 $O(n^2+m)$，n 为 F 中 XSFD 的个数，m 为 $Paths(S)$ 中元素类型路径的个数。

证明：正确性。显然算法是正确的。

时间复杂度。本算法的时间复杂度由 Pathset-strong-clos-ure(X,F)的时间复杂度决定，即为 $O(n^2+m)$。

4 不完全信息环境下的 XML 范式

本节给出不完全信息环境下 XML 范式的定义，通过实例对数据冗余产生的原因进行分析，给出消除数据冗余的转换规则和规范化算法。

定义 12(强键路径) 设 XML Schema $S=(CE,SE,A,D,CP,SP,R,Type,r)$，$Paths(S)$ 为 S 上的路径集合。$X\subseteq Paths(S)$，$y\in Paths(S)$，若 $X\xrightarrow{S}y.@a$ 或 $X\xrightarrow{S}y.D$ 在 S 上成立，$X\xrightarrow{S}y$ 在 S 上也成立，X 为单个路径，称 X 为强键路径，X 为多个路径的集合，称 X 为强键路径集。

定义 13(不完全信息环境下的 XML 范式) 设 XML Schema $S=(CE,SE,A,D,CP,SP,R,Type,r)$，$Paths(S)$ 为 S 上的路径集合，F 为 S 上的 XSFD 的集合。若任意 $X\xrightarrow{S}y.@a\in F$ 或 $X\xrightarrow{S}y.D\in F$，则 X 都为强键路径(集)，称此 S 满足不完全信息环境下 XML 范式，记作(IIC)XNF。

例 1 下面给出一个公司 2008 年职工工资发放情况的 XMLSchema S_1，如图 1 所示。

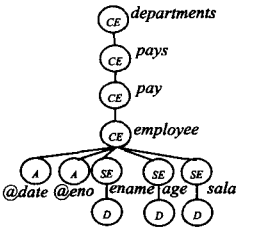

图 1 XML Schema S_1

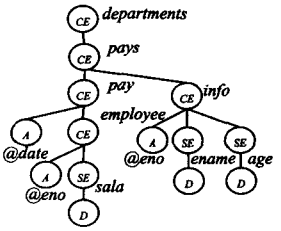

图 2 XML Schema S_2

这一模式表示要存储公司(*departments*)职工(*employee*)的个人信息，如职工号(*eno*)、职工姓名(*ename*)、职工年龄(*age*)以及各个月份(*data*)的工资(*sala*)情况。XML 强函数依赖 $departments.pays.pay.employee.@eno\xrightarrow{S}\{departments.pays.pay.employee.ename, departments.pays.pay.employee.age\}$，$departments.pays.pay\xrightarrow{S}departments.pays.pay.employee.@date$ 在 S_1 上成立，符合 S_1 的不完全 XML 文档树 T_1 如图 3 所示，其中不完全信息的语义为 $\phi_1\doteq\phi_3=\{\text{“}Wang\text{”},\text{“}Zhao\text{”}\}$，$\phi_2\doteq\phi_4=\{\text{“}Xu\text{”},\text{“}Gu\text{”}\}$。

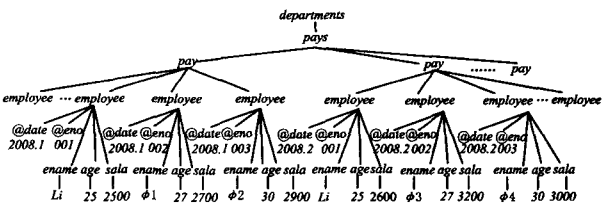

图 3 符合 S_1 的不完全 XML 文档树 T_1

在 T_1 中路径 *deparments.pays.pay* 和 *departments.pays.pay.employee.@eno* 都不是强键路径，所以 S_1 不满足(IIC)XNF，在 T_1 中也的确存在数据冗余，原因是 *@date* 是 *pay* 的属性，但却嵌在了 *employee* 的下面，而职工的自然信息(如 *ename*，*age*)与职工的工资没有直接联系，也嵌在了职工工资的下面，所以 *@date*，*ename* 和 *age* 在 T_1 中存在数据冗余。由于 *@date* 和 *employee* 没有直接的联系，而与 *pay* 有直接联系，所以移动属性 *@date* 为 *pay* 的属性；职工的自然信息与职工的工资没有直接联系，可以采用新建节点的方法，把没有直接联系的信息放在新建节点的下面。S_1 修改后所对应的 XML Schema S_2 如图 2 所示。符合 S_2 的不完全 XML 文档树 T_2 如图 4 所示，ϕ_1，ϕ_2 的意义同上。在 T_2 中路径 *deparments.pays.pay*，*departments.pays.pay.employee.@eno* 和 *departments.pays.info.@eno* 都是强键路径，S_2 满足(IIC) XNF，的确在 T_2 中没有冗余数据。

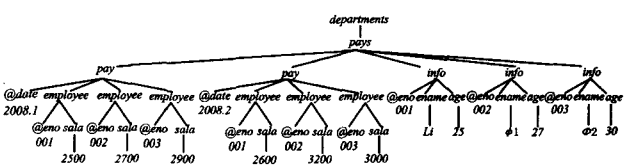

图 4 符合 S_2 的不完全 XML 文档树 T_2

根据上面的实例分析，若复杂类型元素 B 嵌套在复杂类型元素 A 的下面，复杂类型元素 B 带有属性 $@a_1,\cdots,@a_m$

(共有 m 个属性)或简单类型元素 $S_1,\cdots,S_n$(共有 n 个简单元素),而属性 $@a_1,\cdots,@a_m$ 或简单类型元素 $S_1,\cdots,S_n$ 只与复杂类型元素 A 有关系,为了减少数据冗余,移动属性 $@a_1,\cdots,@a_m$ 或简单类型元 $S_1,\cdots,S_n$ 为 A 的属性或子元素,此转换情形如图 5 所示。

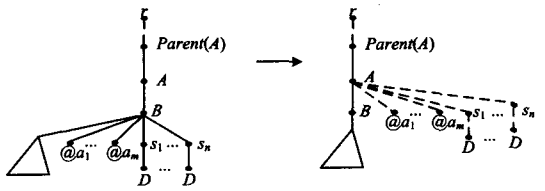

图 5 转换情形 1

根据转换情形 1,下面给出其转换规则。

转换规则 1 移动属性和简单元素所在的子树。

设 S 转换为 $S'=(CE',SE',A',D',CP',SP',R',Type',r')$,其中

$CE':=CE$; $SE':=SE$; $A':=A$; $D':=D$; $r':=r$;

$R'(A):=R(A)\cup @a_1\cup,\cdots,\cup @a_m$;

$CP'(A):=CP(A)\cup S_i\cup,\cdots,\cup S_n$;

$R'(B):=R(B)-\{@a_1\cup,\cdots,\cup @a_m\}$;

$CP'(B):=CP(B)-\{S_i\cup,\cdots,\cup S_n\}$;

$Type'$ 与 $Type$ 的意义相同;SP' 与 SP 的意义相同;

$F':=F-r/\cdots/A\xrightarrow{S}\{r/\cdots/A/B/@a_i,r/\cdots/A/B/S_j/D\}\cup r/\cdots/A\xrightarrow{S}\{r/\cdots/A/@a_i,r/\cdots/A/S_j/D\}(i\in[1,m],j\in[1,n])$。

若复杂元素 B 嵌套在复杂元素 A 的下面,B 下面的属性 $@a_1,\cdots,@a_m$ 或简单类型元素 $S_1,\cdots,S_n$ 与 A 没有直接关系,此时创建新的节点 $info$ 减少冗余,$info$ 的子元素为 $S_1,\cdots,S_n$,属性为 $@a_1,\cdots,@a_m$。转换情形如图 6 所示。

根据转换情形 2,下面给出其转换规则。

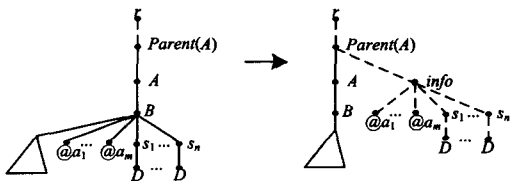

图 6 转换情形 2

转换规则 2 创建复杂元素节点。

设转换 S 为 $S'=(CE',SE',A',D',CP',SP',R',Type',r')$,其中

$CE':=CE\cup info$;$SE':=SE$; $A':=A$; $D':=D$; $r':=r$;

$CP'(Parent(A)):=CP(Parent(A))\cup info$;

$CP'(info):=S_1\cup,\cdots,\cup S_n$;

$R'(info):=@a_1\cup,\cdots,\cup @a_m$;

$CP'(B):=CP(B)-\{S_1\cup,\cdots,\cup S_n\}$;

$R'(B):=R(B)-\{@a_1\cup,\cdots,\cup @a_m\}$;

$Type'$ 在 $Type$ 的意义上加上 $info$ 所对应的数据类型和相应的数据约束;

SP' 与 SP 意义相同;

$F':=F-r/\cdots/A/B/@a_i\xrightarrow{S}r/\cdots/A/B/S_j/D\cup r/\cdots/Parent(A)/info/@a_i\xrightarrow{S}r/\cdots/Parent(A)/info/S_j/D(i\in[1,m],j\in[1,n])$。

定义 14(异常最大化 XSFD) 设 F 为 S 上的 XSFD 的集合,$X\xrightarrow{S}Y\in F$,满足下面的条件,称 $X\xrightarrow{S}Y$ 为异常最大化 XSFD。

1)X 不为强键路径(集);

2)不存在与 1)条件相同的 XSFD 左部路径集。

下面给出规范化算法。

算法 3 Normalization-IICXNF{不完全信息环境下转换 XML Schema 为(IIC)XNF 的算法}

输入:XML Schema $S=(CE,SE,A,D,CP,SP,R,Type,r)$,$S$ 上的 XSFD 集为 F;

输出:满足(IIC)XNF 的 XML Schema $S'=(CE',SE',A',D',CP',SP',R',Type',r')$;

Normalization-IICXNF(S,F)

begin

(1) $S':=S$; //初始化

(2)若 S' 是(IIC)XNF,那么转向(4)。

(3)对 F 中每个异常最大化 XSFD,根据它的不同形式分别处理:

情况 1:如果 $r/\cdots/A\xrightarrow{S}\{r/\cdots/A/B/@a_i,r/\cdots/A/B/S_j/D\}\in F$ $(i\in[1,m],j\in[1,n])$且 Membership$(r/\cdots/A\xrightarrow{S}\{r/\cdots/A/B,r/\cdots/A/B/S_j\},F)=false$,那么利用转换规则 1 转换;

情况 2:如果 $r/\cdots/A/B/@a_i\xrightarrow{S}r/\cdots/A/B/S_j/D\in F(i\in[1,m],j\in[1,n])$)且 Membership$(r/\cdots/A/B/@a_i\xrightarrow{S}r/\cdots/A/B/S_j,F)=false$,分为两种情况:

①若在 F 中不存在某些 XSFD 的左部路径集与$\{r/\cdots/A/B/@a_i\}$有交集,那么利用转换规则 2 转换;

②若在 F 中存在某些 XSFD 的左部路径集与$\{r/\cdots/A/B/@a_i\}$有交集,设交集为$\{r/\cdots/A/B/@a_1,\cdots,r/\cdots/A/B/@a_k\}(k<m)$,那么利用转换规则 2 转换,且保留 $@a_1,\cdots,@a_k$ 仍为复杂元素 B 的属性;

(4)算法结束,输出 S'。

定理 4 算法 3 是正确的,其时间复杂度为 $O(bn^2+bm)$,n 为 F 中 XSFD 的个数,b 为 F 中异常最大化 XSFD 的个数,m 为 $Paths(S)$ 中的元素类型路径个数。

证明:显然算法是正确的。

算法的时间复杂度。算法的时间复杂度由算法 Membership 执行的次数和 Membership 的时间复杂度决定,算法 Membership 的执行次数由异常最大化 XSFD 的个数决定,即为 b,而 Membership 时间复杂度为 $O(n^2+m)$,所以整个算法的时间复杂度为 $O(bn^2+bm)$。

结束语 关于不完全信息环境下 XML Schema 规范化理论,目前还没有查到国内的相应文献。本文研究了此问题,提出了 XML Schema、符合 XML Schema 的不完全 XML 文档树等概念,基于存在型不完全信息的语义提出了 XML 强函数依赖以及相应的推理规则,给出了路径集强闭包和成员籍问题的算法。基于 XML 强键路径(集)给出了不完全信息环境下 XML 范式的定义和相应的规范化算法,此算法减少了不完全信息环境下 XML 文档的数据冗余,实现了更好的 XML Schema 模式设计。在以后的工作中,将对不完全信息环境下存在多值依赖的 XML Schema 的规范化做进一步研究。

参 考 文 献

[1] 吕腾,顾宁,施伯乐. XML DTD的一种范式[J]. 计算机研究与发展,2004,41(4):615-620

[2] 谈子敬,施伯乐. DTD的规范化[J]. 计算机研究与发展,2004,41(4):594-600

[3] 吴永辉. 消除结构冗余的XML数据库模式规范化设计[J]. 软件学报,2004,41(10):1809-1811

[4] Arenas M, Libkin L. A normal form for XML documents [C]// Proceedings of the 21th ACM SIGA-CT-SIG-MOD-SIGART Symposium on Principles of D-atabase Systems. Madison, Wisconsin, USA: ACM Press, 2002: 85-96

[5] Vincent M W, Liu Jixue. Multivalued dependencies and a 4NF for XML[C]// International Conference on Advance in formation Systems Engineering. Klagenfurt, Austria, 2003

[6] Vincent M W, Liu Jixue, Liu Chengfei. A redundancy Free 4NF for XML[C]// The first International XML Database Symposium. Berlin, Germany, 2003

[7] Vincent M W, Liu Jixue, Liu Chengfei. Strong functional dependencies and their application to normal forms in XML[J]. ACM Transactions on Database System, 2004, 29(3): 445-462

[8] 郝忠孝. 空值环境下数据库导论[M]. 北京:机械工业出版社,1996

[9] 殷丽凤,郝忠孝. XML强函数依赖的推理规则[J]. 计算机科学,2008,35(9):165-167

[10] 殷丽凤,郝忠孝. XML强闭包依赖的研究[J]. 计算机科学,2008,35(11):591-594

(上接第171页)

基于FLOC框架,可以将现有的基于执行轨迹的软件缺陷定位方法分成两类:1. 选择一个未通过的测试用例;2. 选择所有未通过的测试用例。但是现实情况下一个软件缺陷可能导致多个测试用例未通过,多个不同的缺陷也可能只引发一个未通过的测试用例。如果能够通过测试用例选择方法将同一个缺陷引发的多个未通过的测试用例选择出来作为一组,那么所有未通过的测试用例将被分成 n 组,理想情况下 n 就是缺陷的个数。如果按组来定位缺陷,那么定位该缺陷的信息将更充分。

现有基于距离的执行轨迹选择方法大都基于代码级的执行轨迹组织方式,执行轨迹的每一维通常对应程序中某行或某几行代码,对于复杂的测试用例执行轨迹,代码级的显然不能很好地描述执行轨迹间的相似性,例如,某个比较复杂的函数包含大量代码,那么计算相似性时,该函数的影响就会很大。可以从代码层向上抽象到函数层或类层,从而提高抽象的粒度,然后通过在函数层上计算执行轨迹间的距离,以避免偏向于复杂的函数。

目前计算怀疑率的方法都是分别计算每一个语句块的怀疑率,而没有考虑语句块间的交互。实际情况下语句块与语句块间的交互可能更好地体现软件中的缺陷,可以通过数据挖掘的方法挖掘执行轨迹中的这些交互模式(语句块的某种组合),将缺陷定位到某些语句块的组合。

结束语 本文给出了基于执行轨迹的软件缺陷定位方法的通用框架—— FLOC,并详细介绍了该框架的各个阶段。把现有基于执行轨迹的缺陷定位方法分解成框架下各个子方法的组合,综合比较了现有方法,并提出了改进思路。

目前软件缺陷定位技术还需要拓展和深化,例如,如何在海量数据中提高缺陷定位的精度;如何在数据量有限的情况下(例如系统运行过程中通过网络返回的失效相关数据)定位缺陷;如何利用已有的数据挖掘技术进行缺陷模式的挖掘等问题都需要进一步研究。

参 考 文 献

[1] Agrawal H, Horgan J, London S, et al. Fault localization using execution slices and dataflow tests[C]// Proceedings of IEEE Software Reliability Engineering. Los Alamitos CA: IEEE Computer Society Press, 1995: 143-151

[2] Renieris M, Reiss S P. Fault Localization with Nearest Neighbor Queries[C]// International Conference on Automated Software Engineering. Los Alamitos CA: IEEE Computer Society Press, 2003: 30-39

[3] Jones J A, Harrold M J, Stasko J. Visualization of Test Information to Assist Fault Localization[C]// Proceedings of International Conference in Software Engineering. Los Alamitos CA: IEEE Computer Society Press, 2002: 467-477

[4] Stoerzer M. Finding Failure-Inducing Changes in Java Programs using Change Classification[C]// The 14th ACM Symposium on Foundations of Software Engineering. Oregon, USA, 2006

[5] Groce A D. Error Explanation and Localization with Distance Metrics[D]. Pittsburgh, PA: Carnegie Mellon University, 2005

[6] Hao Dan, Pan Ying. A Similarity-Aware Approach to Testing Based Fault Localization[C]// International Conference on Automated Software Engineering. California, USA, 2005

[7] Chen T Y, Cheung Y Y. On program dicing[J]. Software Maintenance: Research and Experience, 1997, 9(1): 33-46

[8] Ren Xiaoxia, 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

[9] Jones J A. Fault Localization using Visualization of Test Information[C]// International Conference in Software Engineering. Edinburgh, UK, 2004

[10] Cleve H. Locating Causes of Program Failures[C]// International Conference in Software Engineering. Missouri, USA, 2005

[11] Zeller A. Isolating cause-effect chains from computer programs [C]// Tenth ACM SIGSOFT Symposium on the Foundations of Software Engineering(FSE-10). South Carolina, USA, 2002

[12] Dickinson W, Leon D, Podgurski A. Pursuing failure: The distribution of program failures in a profile space[C]// Proceedings of 9th ACM SIGSOFT Symposium on the Foundation of Software Engineering. 2001: 246-255

[13] Harrold M J, Rothermel G, Sayre K, et al. An empirical investigation of the relationship between spectra differences and regression faults[J]. Software Testing, Verification and Reliability, 2000, 10(3): 171-194

[14] Jones J A. Empirical Evaluation of the Tarantula Automatic Fault Localization Technique[C]// The 20th International Conference on Automated Software Engineering. New York: ACM Press, 2005: 273-282