计算机科学2004Vol. 31№. 9A

基于树自动机的 XQuery 子类型关系和值-域关系

谢荣传[1] 谢铉洋[2]

(安徽大学计算智能与信号处理教育部重点实验室 合肥230039)[1]

(中国科学技术大学计算机科学与技术系 合肥230026)[2]

摘 要 XQuery 是 W3C 提出的一种对 XML 结构文档或数据进行查询的语言规范,该语言规范对其类型系统仅做了功能性的描述,而没有涉及在编译实现中遇到的两个类型之间子类型关系和值-域关系的判断方法。本文描述和分析了这两种关系,基于树自动机,给出了 XQuery 类型系统中子类型关系和值-域关系的判断算法。

关键词 Xquery,类型系统,树自动机

Tree Automata Based Algorithms for Sub-typing and InDOM Relationships in XQuery

XIE Rong-Chuan[1] XIE Xuan-Yang[2]

(Key Laboratory of Intelligent Computing & Signal Processing, Anhui University, Hefei230039)[1]

(Department of Computer Science and Technology, University of Science and Technology of China, Hefei 230027)[2]

Abstract Proposed by W3C, XQuery is a descriptive query language for XML structured documents or data. The specification of XQuery gives functional description of XQuery's typing system, but doesn't provide the way to judge th sub-typing relationships and inDOM relationships in language compiling. In this paper, the two important relations: sub-typing relationships and inDOM relationships are described and analysed. After some definitions of Tree Automaton, the judgment algorithms for this two kinds of relationships are presented.

Keywords XQuery, Typing system, Tree automata

1 问题描述

XQuery 是 W3C 提出的一种对 XML 结构文档或数据进行查询的语言规范,在该语言的实现中,由于它是一种描述性的语言,即用户无需显式说明数据类型而由系统进行自动推断,给该语言的实现带来了一些困难,因为我们需要有一套完整的类型系统来支持该语言的运行。

XQuery 的类型系统支持各种正规表达式运算符,这些运算符如表1所示。这些算子任何有意义的组合运算在 XQuery 中都是允许的,比如类型:name,mail*,tel? 分别是类型 name,mail*,tel* 的子类型,因为凡是符合前者类型定义的数据一定同时也符合后者的类型定义。子类型的严格定义在下一节给出。

表1 XQuery 支持的算子

类型	算子
重复	+ ? *
连接	,&
选择	|
括号	()

查询在被执行时,常见的一种操作是根据当前某个变量值的不同情况进行分支选择。比如:

```
typeswitch Person
    case (name,mail*,tel)://如果 Person 值中
        含有 tel 元素
    case (name,mail*):// 如果 Person 值中不含
        有 tel 元素
```

该操作的本质是进行值与类型的匹配关系判断,比如上面第一个 case 分支用于判断 Person 值是否与类型(name,mail*,tel)匹配。这种匹配关系称为值-域关系。

由于 XML 自身层次嵌套和 XQuery 中上述正规性算子的存在,使得 XQuery 的类型系统结构相当复杂。本文从树自动机的角度给出上述两种关系的判断算法,利用自动机的正规性来解决 XQuery 类型与值的嵌套处理。

2 子类型关系和值-域关系

两个类型之间的子类型关系是 XQuery 处理器在进行类型检查和查询结果类型分析中常常用到的一种关系判断。假设有两个类型定义 ROOT$_1$,

谢荣传 副教授,主要研究方向:数据库、Internet 应用、多媒体技术;谢铉洋 博士生,主要研究方向:信息融合。

$ROOT_2$，若一个 XML 元素 elm 满足 $ROOT_1$的定义，同时也满足 $ROOT_2$的定义；反之亦然。这个过程可以记为：

$$ROOT_1 <: ROOT_2 \wedge ROOT_2 <: ROOT_1 \rightarrow ROOT_1 = ROOT_2。$$

下面我们通过树自动机来给出 XQuery 类型系统中两种关系的严格定义。树自动机(Tree Automata)，简称 TA[2]，与编译理论中一般的自动机[3](DFA，NFA)类似，也是根据状态进行转移的自动机，所不同的是 TA 将自动机组织成树的形式而不是图。有关 TA 的演示可以参看文[4]。

定义1(TA 的语言及 Acceptance)　一个语言 L 可以被树型自动机 TA 接受(Acceptance)是指，该语言的所有可能的输入 t 都可以被 TA 接受，即满足：

$$A: t \xrightarrow{*} qf$$

并记该语言 $L=L(TA)$。

定义2(inclusion)　设定义在相同符号集上的两个 TA

$A1=(Q1,F,Qf1,\Delta1)$，$A2=(Q2,F,Qf2,\Delta2)$

则称 A1和 A2满足 inclusion 关系的充分必要条件是：

$$\forall\, t\in L(A_1)\Rightarrow A2: t \xrightarrow{*} q$$

从 inclusion 的定义中可以看出：如果 A1和 A2满足 inclusion 关系，则能够被 A1接受的语言一定能被 A2接受；反之则不一定。这时记 inclusion(A1，A2)=true，但

inclusion(A2，A1)=false

定义3(equivalence)　如果 inclusion(A1，A2)与 inclusion(A2，A1)成立，则有 equialence(A1，A2)成立。

XQuery 中的类型系统主要由两部分组成：类型关系和值-域(value-domain)关系。其中类型关系包括子类型，类型等价关系判断。

定义4(值-域关系 inDOM)　如果值 v 能够被类型 T 对应的 TA 所接受(acceptance)，则称值-域关系 inDOM(v，T)成立。

定义5(子类型关系 subType)　如果所有满足 inDOM(v，T1)的值 v 同时也满足 inDOM(v，T2)，则称 T1是 T2的子类型，记作 T1<：T2；即 T1<：T2的充要条件是：

$$\forall\, v(inDom(v,T1))\Rightarrow inDom(v,T2)$$

定义6(类型等价)　若 T1<：T2且 T2<：T1则称类型 T1和 T2等价，记为 T1=T2。

3　XQuery 到 TA 的转换函数

考察值-域关系及子类型关系的定义，可以发现它们与 TA 中的相关定义有着内在的联系。但是 XQuery 的类型系统并不与 DFTA(有限状态树自动机)直接对应，因此需要提供某种映射函数 M 将 XQuery 的类型表达式与 DFTA 建立一一映射：

M：XQueryType→DFTA；

需要对映射后的 DFTA 进行 acceptance 和 inclusion 关系的判断以完成 inDOM 和子类型关系的计算。转换函数 M 分两个步骤进行：XQueryType→External→Internal(DFTA)。

3.1　从 XQuery 到 External

首先说明 XQuery 的类型说明的特点：

1. 正规性(regularity)。从 XQuery 的类型定义可以看出它提供选择'|'，连接','，重复'*'和加括号这四种运算。

2. 扩展的正规性(Ex_Regularity)。定义中还包括"+"，"?"，"&"这三个算子。其中"&"的定义为：T1&T2=T1，T2|T2，T1。

3. 名称通配符的支持。该类型定义还支持通配符"*"表示的类型名称。

Haruo Hosova 在文[5，6]中，提出了利用 TA 解决子类型关系判断的方法。但是在他的方法中并没有对"&"算子，类型通配符和 XML 属性定义提供支持，而这三个新特性的引入，极大地扩展了类型表达式的表达能力。

下面首先定义 External 格式，然后给出从 XQuery 说明到 External 的转换方法。

定义7(External 格式)　类型说明的 External 格式是一个单根的树 T

T::=	()空序
	label {T}　元素定义
	Δlabel {T}　属性定义
	T，T　","算子，连接运算
	T|T　"|"算子，选择运算
	T&T　"&"算子，插入运算
	(T)　括号运算
	x　元素类型变量
	Δx　属性类型变量

形式化地说，上面定义了一个有限符号集 L，一个有限变量集合 X 和一个变量到定义的类型的映射集合 E：

$$L:\{label\cup\Delta label\}$$

$$x:\{x\cup\Delta x\}$$

$$E:\{\chi\rightarrow\tau\mid\chi\in X\wedge\tau\in T\}$$

作为特殊的情况，可以将基本类型比如 xs：int，xs：string 等看作是 int{()}和 string{()}；这样可以认为有下面的关系成立：1 inDOM int{()}。

注意到 external 的定义中允许递归地出现，而且递归变量的位置是不加以限制的。因为判断一个上下文无关文法是不是正规(注意这里的正规是指

文法而言，而不是上面的正规性）是不可判定的[3]，所以在实现中需要对上面的文法递归加以限制，即只允许递归是“右线性”的（即消除左递归），幸运的是，这种限制在实际的应用中很少会被突破。

表2中我们给出从 XQuery 的类型说明到 External 的转换规则。

表2 XQuery 到 External 的转换规则

	当前 XQuery 输入	External 输出		当前 XQuery 输入	External 输出
[01]	define element QName { Type? }	T=QName { Type } |T=QName {()}|	[10]	“type”QName	X=QName
[02]	define attribute QName { Type? }	T=QName { Type } |T=ΔQName {()}|	[11]	“(“Type ? ”)”	按照 Type 递归处理或()
[03]	define type QName { Type? }	T=XX=Type|()	[12]	“none”	X=()
[04]	Type“|” Type	T=Type | Type	[13]	QName	T=QName { ()}表示简单的基本类型
[05]	Type “&”Type	T=Type & Type	[14]	“attribute” QName	X=QName
[06]	Type “,” Type	T=Type,Type	[15]	“attribute” NameTest { Type ? }	X=_{ Type } | ⊥{ ()}
[07]	Type*	T=X X=Type,X | ()	[16]	“element” Qname	ΔX=QName
[08]	Type+	T=Type,X X=Type,X | ()	[17]	“element” NameTest { Type ? }	ΔX=_{ Type } | ⊥{ ()}
[09]	Type ?	T=X X=Type | ()			

3.2 从 External 到 Internal

定义8（Internal 格式） 内部格式（Internal Form）是如下一组转换规则。

A ::=∈ 叶转换
label(S,S) 元素枝转换
Δlabel(S,S) 属性枝转换
A|A 选择
ϕ 空转换

形式化地说，上面定义了：

状态转换集合 $A=\{A_i \mid A_i \in$ 转换规则$\}$；

状态集合 $S=\{S_i \mid S_i \in$ 状态$\}$；

映射集合

$$M=\{(S_i \to A_i) \mid S_i \in S \wedge A_i \in A\};$$

上面定义的语义为：

a ::= ∈
label(a,a′)
Δlabel(a,a′)

∈代表叶节点，即状态转换到此结束。

label(a,a′)表示元素(label)，它的第一个项（属性或元素）的类型说明对应 a，而该元素的剩余部分定义对应于 a′。

Δlabel(a,a′)表示属性 label，该属性的类型对应 a，而剩余的部分由 a′对应。

事实上，上面定义的 Internal Form 就是一个二元的树自动机 $A=(Q,F,Q_f,\Delta)$，其中：

$$\begin{cases} Q=S \\ F=\{label(,),\Delta label(,),\in \mid arity(label)=arity(\Delta label)=2 \wedge arity(\in)=0\} \\ Q_f=\{so\} \\ \Delta=M \end{cases}$$

如果我们能从 XQuery→External Form→Internal Form 进行转化，然后利用 TA 理论中的 inclusion 属性就可以解决子类型关系<：的判断；并且如果一个给定的值，则有 v∈L(TA)，则有 v in Dom TA 代表的类型。

下面给出从 External 格式到 Internal 的转换规则，该规则由 ts(External)=Internal 函数表示：

$$ts(())=\epsilon \quad (1)$$
$$ts(label\{T\})=label(S_T,S_0) \quad (2)$$
$$ts(\Delta label\{T\})=\Delta label(S_T,S_0) \quad (3)$$
$$ts(T_1|T_2)=ts(S_{T1})|ts(S_{T2}) \quad (4)$$
$$ts(T_1\&T_2)=ts(T_1,T_2)|ts(T_2,T_1) \quad (5)$$
$$ts((),T)=ts(T) \quad (6)$$
$$ts(label\{T_1\},T_2)=label(S_{T1},S_{T2}) \quad (7)$$
$$ts(\Delta label\{T_1\},T_2)=\Delta label(S_{T1},S_{T2}) \quad (8)$$
$$ts(X)=ts(E(X)) \quad (9)$$
$$ts(\Delta X)=ts(E(\Delta X)) \quad (10)$$
$$ts(X,T)=ts(E(X),T) \quad (11)$$
$$ts(\Delta X,T)=ts(E(\Delta X),T) \quad (12)$$
$$ts((T_1,T_2),T_3)=ts(T_1,(T_2,T_3)) \quad (13)$$

$$ts((T_1|T_2),T_3)=ts(T_1,T_3)|ts(T_2,T_3) \quad (14)$$

$$ts((T))=ts(T) \quad (15)$$

说明：

·上面公式中 S_T 表示对应于 External 类型 T 的 Internal 类型 $A_{S_T}=ts(T)$ 的状态。

·对每一个给定的 External 类型定义 T，经过上面的转换规则后，我们将得到一个 Internal 的类型表示 A 和一个上下文环境 Π。初始 Π=Φ，计算从 A0=ts(X0)Φ 开始，在计算的每一个步骤中，将每一个新出现的状态 S_T 和它所对应的 Internal 类型 A_{S_T} 之间的映射添加到 Π 中。

当没有新的状态 S 出现，即计算过程所出现的所有 S∈Π 时，则转换结束。限于篇幅，详细的例子请参考文[7]。

4 SubType 算法

XQuery 的类型说明转换为 Internal 格式后，SubType 算法就可以直接在 TA 的基础上给出。在给出算法之前，首先处理一些算法的特殊情况或称边界条件。

经过转换最终得到了一个对应于 XQuery 类型定义的 TA。而子类型关系的判断在上文已说明为两个 TA 之间的 inclusion 关系的判断，所以这里直接给出两个 TA 之间的 inclusion 关系判断算法。

这里，由于 XQuery 自身定义的复杂性，我们发现有如下的难点所在：

·由于属性的特点规定属性之间的顺序是无关紧要的，因此，对于 Δ 符号（即属性的定义）要作特殊的处理；

解决方法：在开始计算之前作一个预处理的工作，任务就是将属性按照字母顺序排序。

·对简单类型和通配类型的支持，这些通配的类型在 TA 中都是以 label(ε,ε)的形式存在的，具体来说就是如下7种形式：xs:anyType(ε,ε)，xs:anySimpleType (ε, ε)，xs: anyComplexType (ε, ε)，xs: anyElement (ε, ε)，xs: anyAttribute (ε, ε)，xs: anyItem(ε,ε)，xs:anyNode(ε,ε)。

下面给出子类型关系的判断算法，该算法完全建立在 TA 的基础上，是 TA 中 inclusion 关系的直接扩展。其中 Γ 表示子类型关系环境，它存储了当前所有已知的子类型关系，Γ′ 及 Γ″ 等表示添加了新的子类型关系之后的扩展类型环境。

$$\frac{(A<:B)\in\Gamma}{\Gamma a\ A<:B\Rightarrow\Gamma} \quad (1)$$

$$(A<:B)\notin\Gamma$$

$$\frac{\Gamma;(A<:B)a^*A<:B\Rightarrow\Gamma'}{\Gamma a\ A<:B\Rightarrow\Gamma'} \quad (2)$$

$$\frac{}{\Gamma a^*\Phi<:A\Rightarrow\Gamma} \quad (3)$$

$$\frac{\Gamma a^*A<:B\Rightarrow\Gamma' \quad \Gamma' a^*A'<:B\Rightarrow\Gamma''}{\Gamma a^*A|A'<:B\Rightarrow\Gamma''} \quad (4)$$

$$\frac{A=\varepsilon|A'}{\Gamma a^*\varepsilon<:A\Rightarrow\Gamma} \quad (5)$$

$$C=label'(B,B')|C'$$

$$canPrune(label,label')=true$$

$$\frac{\Gamma a^*label(A,A')<:C\Rightarrow\Gamma'}{\Gamma a^*label(A,A')<:C\Rightarrow\Gamma'} \quad (6)$$

$$C=\varepsilon|C'$$

$$\frac{\Gamma a^*label(A,A')<:C'\Rightarrow\Gamma'}{\Gamma a^*label(A,A')<:C\Rightarrow\Gamma'} \quad (7)$$

表3 canPrune 函数的剪枝条件

Label1 / Label2	基本数据类型	通配类型	元素定义	属性定义
基本数据类型	按照表4返回结果，如果 label1 在 label2 的下方或相同，则返回 true；否则返回 false.	lable2＝anyType anyComplexType，anySimpleType 则返回 true；否则返回 false.	false	false
通配类型	false	查询表4，如果对应位置是1则返回 true；否则返回 false.	false	false
元素定义	false	label2＝anyType，anyComplexType，anyElement 则返回 true；否则返回 false.	label1 ＝ label2 则返回 true；否则返回 false.	false
属性定义	false	label2＝anyType，anyComplexType，anyAttribute 则返回 true；否则返回 false.	false	label1＝label2 则返回 true；否则返回 false.

注：

(1)将上面的 label 换成 Δlabel 仍然是成立的。

(2)注意到 XQuery 的简单类型之间是有层次的，该层次就是 XQuery 基本类型图。

(3)在规则中，当右边的转换是 xs:anyType 时规则仍然是成立的。

(4)在规则中,假设有函数 canPrune(label1,label2),该函数的功能见表3。当查询对应的单元为 false 时才剪支。

5 值-域关系算法

值-域关系即 inDOM 关系,它在类型分析时用不到,但是在值计算时常被使用,常用来判断一个 XML 元素、属性等是否能够匹配一个类型从而可以选择 core 语法树的不同分支计算。由于其与树自动机的关系相当密切,因此在本节给出该关系的判断算法。判断算法直接在 TA 的基础上给出而不是在 XQuery 类型说明之上。

在算法说明中,符号$\subset:$表示 inDOM 关系,$\vee$表示或关系,Π 为 internal 的自动机状态转换映射:

$$\frac{true}{\Phi\subset:T} \quad (1)$$

$$\frac{true}{\varepsilon\subset:\varepsilon\mid T} \quad (2)$$

$$\frac{v\subset:T_1\vee v\subset:T_2}{v\subset:T_1\mid T_2} \quad (3)$$

$$\frac{v\subset:T_1}{v\subset:T_1\mid\varepsilon\wedge v\neq\varepsilon} \quad (4)$$

$$\frac{element.name=label\wedge element.childNodes\subset:\Pi(S_L)\wedge\varepsilon\subset:\Pi(S_R)}{element\subset:label(S_L,S_R)} \quad (5)$$

$$\frac{elmList[1].name=label\wedge elmList[1]\subset:label(S_L,\varepsilon)\wedge elmList[REST]\subset:\Pi(S_R)}{elmList\subset:label(S_L,S_R)} \quad (6)$$

$$\frac{elmWithAttr.name=label\wedge attrList+childNodes\subset:\Pi(S_L)\wedge\varepsilon\subset:\Pi(S_R)}{elmWithAttr\subset:label(S_L,S_R)} \quad (7)$$

$$\frac{attList[1].name=label\wedge attrList[1]\subset:\Delta label(S_L,\varepsilon)\wedge attrList[REST]\cup elmList\subset:\Pi(S_R)}{attrList,elmList\subset:\Delta label(S_L,S_R)} \quad (8)$$

$$\frac{attr.name=label\wedge attr.value\in\Pi(S_L)}{attr\subset:\Delta label(S_L,\varepsilon)} \quad (9)$$

$$\frac{textNode.vale\in label\wedge\Pi(S_L)=\varepsilon\wedge\Pi(S_R)=\varepsilon}{textNode\subset:label(S_L,S_R)} \quad (10)$$

注:

(1)算法中,使用 name 来获得一个元素或属性的名字,使用 value 来或得一个属性的值,这是可以的,因为属性只能是简单的类型而不能带有结构;

(2)借用了集合符号$\in$来表示一个属性的值是某种类型的。例如:200$\in$xs:int(S0,S0);

(3)注意到(6),如果元素列表只有一个元素,则它退化为(5)。证明是简单的:注意到如果 elmList 只有一个元素时,elmLis[1].name=label 与 element.name=label 等同;elmList[1] inDom Π(SL,ε)=element inDom Π(SL,ε);

φ inDom Π(SR),这部分显然为真;

(4)这里有一个问题是如何对 textNode 节点进行判断,实际上就是算法的结束判断,这就是公式(10)所起的作用;

(5)上面我们还规定了 $\Pi(\in)=\in$;

(6)其他的情况均表示失败;

(7)我们在文[7]中给出了一个详细的算法实例。

结束语 本文给出的子类型关系和值-域关系判断算法已经在我们所实现的 XQuery 查询语言原型系统 NEPTUNE[8]中实际使用,通过文[12]中所提供用例的测试验证了其正确性。文[7]中给出了上面所有算法的实现技术和相应的示例,文[8]则是从算法实现的技术角度进行说明并给出其他的一些相关资料。TA 中 inclusion 关系判断的时间复杂度是指数阶的[2],因此需要从实践的角度对算法实现作一些代码级别的优化。作为整个 XQuery 语言的一部分,这种优化技术包括如文[9]中所给出的缓冲策略,代数优化[10,11]等。

参 考 文 献

1 World Wide Web Consortium. XQuery 1.0:An XML Query Language[EB/OL]. http://www.w3.org/TR/xquery/.

2 Comon H,et al. Tree Automata Techniques andApplications[EB/OL]. http://www.grappa.univ-lille3.fr/tata/tata.pdf

3 Hppcroft J E,Ullman J D. Introduction to Automata Theroy,Lanauages,and Computation[M]. Addison-Wesly,1979

4 Lceland B,et al. TAJA,Tree Tuple Automata for Java,[EB/OL]. http://www.univ-orleans.fr/SCIENCES/LIFO/Members/lecland/taja.php

5 Hosoya H,et al. Regular Expression Pattern Matching for XML[J]. http://citeseer.nj.nec.com/388107.html

6 Hosoya H,et al. XDuce:An XML Processing Language[J]. http://citeseer.nj.nec.com/hosoya99xduce.html

7 谢铉洋. XML:查询语言 XQuery 的编译实现[D]:[硕士学位论文]. 合肥:安徽大学计算机系,2003

8 谢铉洋. XQuery 实现技术报告[R]. 合肥:安徽大学计算机系,2002

9 Jacob A,Leary O,et al. XCache:An XML Query Caching System[EB/OL]. http://davis.wpi.edu/dsrg/XCache2001/

10 McHugh J,Widom J. Query Optimization for XML[J]. In:Proc. of the 25th VLDB Conference. http://www.acm.org/sigmod/vldb/conf/1999/P32.pdf

11 McHugh J,Widom J. Query Optimization for Semistructured Data[R]. http://www-db.stanford.edu/lore/pubs/qo.pdf

12 World Wide Web Consortium. XML Query Use Cases[EB/OL]. http://www.w3.org/TR/xmlquery-use-cases/