一种 Active XML 模式重写算法

马海涛[1] 朱 燕[2] 郝忠孝[1,3]

(哈尔滨工业大学计算机科学与技术学院 哈尔滨 150001)[1]

(燕山大学信息科学与工程学院 秦皇岛 066004)[2]

(哈尔滨理工大学计算机科学与技术学院 哈尔滨 150080)[3]

摘 要 基于树自动机理论,研究了 Active XML(简记为 AXML)模式重写问题,提出了一种多项式时间的 AXML 模式重写判定算法,并对算法进行了实现。实验结果证明了所提算法用于判定 AXML 模式重写的优越性。

关键词 Active XML,Active XML 模式,树自动机,模式重写

中图法分类号 TP393

Algorithm for Rewriting Active XML Schema

MA Hai-tao[1] ZHU Yan[2] HAO Zhong-xiao[1,3]

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

(College of Information Science and Engineering, Yanshan University, Qinhuangdao 066004, China)[2]

(College of Computer Science and Technology, Harbin University of Science and Technology, Harbin 150080, China)[3]

Abstract The problem of schema rewriting is a fundamental problem of Active XML(AXML for short) data exchange and usually has a higher complexity. We defined an extended tree automaton, AXML schema tree automaton(ASTA), which can efficiently describe the set of all AXML documents that conform to the given schema. Then based on ASTA automata, we proposed an algorithm performed in polynomial time for deciding whether one AXML schema can rewrite into the other one. Finally, the experimental results prove that our algorithm for checking AXML schema rewriting can be more efficiency than existing algorithm.

Keywords Active XML, Active XML schema, Tree automata, Schema rewriting

1 引言

Active XML(AXML)作为一种分布式 Web 数据管理工具,由 Serge ABiteBoul 等人于 2002 年首次提出[1]。AXML 文档是一部分数据直接给出、另一部分数据以 Web 服务调用的方式隐含给出的 XML 文档,通过激活这些服务调用,可以获得它们所表示的内涵数据信息。XML 文档中引入嵌入式 Web 服务,增加了文档的动态性和灵活性。然而,也带来了诸多新问题,如文档重写[2]、AXML 文档查询[3]、文档包含及服务调用可终止性问题[4]等。

AXML 模式重写问题是指在进行 AXML 文档数据交换时,发送方需要检验所有符合给定模式的 AXML 文档通过触发服务调用是否能够转换为目标模式的文档实例,这需要对不同模式进行比较。本文基于树自动机理论,定义了用于抽象 AXML 模式的树自动机——ASTA 机(AXML Schema Tree Automata),提出了一种多项式时间的 AXML 模式重写判定算法,并对算法进行了正确性证明和实验分析。

2 基本知识

2.1 AXML

AXML 文档可用包含数据节点和函数节点的有序标签树来表示,其中数据节点描述文档中的普通节点,函数节点描述服务调用。假设存在下列不相交的域:L 表示标签集合,F 表示函数名称集合,D 表示数据值集。

定义 1(AXML 文档)[2] AXML 文档定义为表达式(T, λ),其中 $T=(N,E,<)$为一有序树,N 表示有限节点集合,$E \subseteq N \times N$ 为边集合,$<$符号关联每一节点与其孩子节点的全序关系,$\lambda: N \to L \cup F \cup D$ 表示节点的标签函数,只有叶子节点被赋予 D 中的数据值。

标签属于 $L \cup D$ 的节点称为数据节点,标签属于 F 的节点称为函数节点,称函数节点的孩子子树为函数参数。当函数节点被调用时,将参数传递给函数节点,用函数的返回结果替换文档中的函数节点,称得到的新文档可由原文档生成。

图 1 给出了一个 AXML 文档 t,图中函数节点以矩形节点表示,数据节点由双引号表示。例如,当函数节点 *GetTemp* 以节点 *city* 为参数被调用后,返回结果为一个以节点 *temp* 为根节点的子树,替换掉该函数节点,可生成如图 2 所示文档 t'。

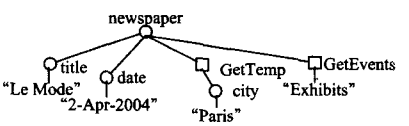

图 1 AXML 文档

到稿日期:2008-06-26 本文受黑龙江省自然科学基金(F2006-01)资助。

马海涛(1977—),男,博士,主要研究领域为数据库理论,E-mail:mahaitao@hit.edu.cn。

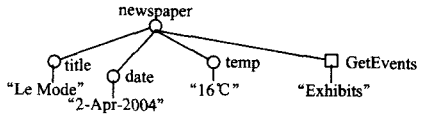

图 2　调用函数之后的 AXML 文档

AXML 模式定义了 AXML 文档规范。为了描述上的简便，考虑一个简单的类似于 DTD 的 AXML 文档模式。

定义 2(AXML 模式)[2]　AXML 文档模式定义为表达式(L,F,τ)，其中 L 和 F 分别为有限标签集合和函数名称集合，τ 为函数，映射标签 $l\in L$ 为 $L\cup F$ 上的正则表达式或关键字"*data*"(表示文本数据)。映射任意函数名称 $f\in F$ 到一对正则表达式 $\tau_{in}(f)$ 和 $\tau_{out}(f)$，分别称为 f 的输入类型和输出类型。

下面给出图 1 所示文档 t 对应的 AXML 模式。

例 1　AXML 模式 $s=(L,F,\tau)$定义。

数据元素：

$\tau(newspaper)=title.date.(GetTemp|temp).GetEvents$,

$\tau(title)=data,\tau(date)=data,\tau(temp)=data$,

$\tau(city)=data,\tau(exhibit)=title.(GetDate|date)$

函数元素：

$\tau_{in}(GetTemp)=data,\tau_{out}(GetTemp)=temp$,

$\tau_{in}(GetEvents)=data$,

$\tau_{out}(GetEvents)=(exhibit|performance)^*$,

$\tau_{in}(GetDate)=data,\tau_{out}(GetDate)=date$

给定 AXML 文档 t，其文档模式 s 定义了该文档通过调用嵌入函数节点可以生成的全部文档集合。

由于函数调用的返回结果中可能含有新的函数节点或已经调用的函数节点，而这将导致无穷计算，因此在对 AXML 文档中函数节点调用之前，必须对函数调用的可终止性进行判定。Serge ABetiBoul 等人在文献[4]中指出，AXML 服务调用终止性问题在约束条件下可多项式时间完成判定。本文的研究重点是 AXML 模式重写判定问题，不考虑文档中函数调用的可终止性问题，所以总是假设文档中的函数调用为可终止调用。

2.2　树自动机理论

树自动机理论随着 XML 的出现重新引起了研究者们的重视，在 XML 中的应用也十分广泛。Neven Frank 在文献[5,6]中综述了自动机、逻辑和 XML 之间的关系。文献[7]从正则树文法的角度，对当前应用比较广泛的 XML 模式语言——DTD，XML Schema 和 RELAX NG 进行了分析和对比，给出了相应的 XML 文档有效性检验算法。在文献[8,9]中，作者将树自动机用于优化 XPath 查询和解决 XML 数据流上高效执行大量 XPath 查询问题。

树自动机是对传统字符自动机的扩展，转移规则定义支持状态集合，这种特性使得树自动机更适合处理树形数据结构。下面回顾无排列树自动机的定义。

定义 3(树自动机)[6]　树自动机定义为 4-元组 $B=(Q,\Sigma,\delta,A)$，其中 Q 为有限状态集合，Σ 为有限字母表，$A\subseteq Q$ 表示终止状态集合，δ 为映射函数 $Q\times\Sigma\to 2^{Q^*}$，对于任意 $a\in\Sigma$ 和 $q\in Q$，$\delta(q,a)$为定义 Q^* 在上的正则表达式，通常采用有限状态机表示。树自动机 B 定义的语言为正则树语言，记作 $L(B)$。

3　AXML 模式重写判定算法

本文所解决的 AXML 模式重写判定问题描述如下：给定 AXML 源模式 s 及目标模式 s'，判定符合模式 s 的所有 AXML 文档是否能够全部重写为目标模式 s'的文档实例。

3.1　ASTA 机

根据树自动机理论，DTD 模式可以用无排列树自动机来表达。DTD 对应的树自动机所定义的语言，对应了满足 DTD 约束的 XML 文档集合。AXML 模式是对 DTD 的一个扩展，基于同样理论基础，本文采用无排列树自动机来抽象 AXML 文档模式，称之为 ASTA 机。令该类树自动机定义满足 AXML 模式约束的文档集合。

定义 4(ASTA 机)　ASTA 机定义为 4-元组 $B=(Q,\Sigma,\delta,A)$，其中 Q 为有限状态集合，$\Sigma=\Sigma_L\cup\Sigma_F$($\Sigma_L$ 和 Σ_F 分别为元素标签和函数元素名称的有限集合)，$A\subseteq Q$ 为终止状态集合，转移函数 δ 分为两部分：$\delta_{data}:Q\times\Sigma_L\to 2^{Q^*}$ 和 $\delta_{function}:Q\times\Sigma_F\to 2^{Q^*}$，对于任意 $q\in Q,a\in\Sigma_L$，满足 $\delta(q,a)$为定义在 Q^* 上的正则语言；对于任意 $q\in Q,f\in\Sigma_F$，满足 $\delta(q,f)$为定义在 Q^* 上的一对正则语言$(\delta_{in},\delta_{out})$，分别定义了函数元素 f 的输入和输出类型实例集合。

由于 W3C 组织定义 XML 类型规范时指出，XML 文档规范 DTD 或 XML Schema 定义要求其内容模型必须是确定性的，或称为无二义性[10]。这为定义 AXML 模式提供了参考方案，因为给定任意一个无二义性的正则表达式，可多项式时间构造一个等价的确定性 Glushkov 自动机。因此，在 ASTA 机定义中，本文采用确定性有限自动机(Deterministic Finite Automata，DFA)来描述转移函数定义。在下文的 ASTA 机定义中，为了描述方便，采用 δ 来表示 δ_{data}，而函数元素的转移函数直接采用 DFA 对(M_{in},M_{out})表示。下面以例 1 中的 AXML 模式为例，给出模式 s 所对应的 ASTA 机。

例 2　ASTA 机 $B=(Q,\Sigma,\delta,A)$定义

$Q=\{q_0,q_1,q_2,q_3,q_4,q_5,q_6,q_7,q_8,q_9,q_{10}\}$,

转移函数 δ 定义：

$\delta(q_1,title)=\varepsilon,\delta(q_2,date)=\varepsilon$,

$\delta(q_3,temp)=\varepsilon,\delta(q_4,city)=\varepsilon$,

$\delta(q_5,performance)=\varepsilon,\delta(q_8,exhibit)=q_1.(q_7|q_2)$,

$\delta(q_7,hotel)=q_1.q_2.q_3.q_6$,

$\delta_{in}(q_6,GetTemp)=q_4,\delta_{out}(q_6,GetTemp)=q_3$,

$\delta_{in}(q_7,GetDate)=q_1,\delta_{out}(q_7,GetDate)=q_2$,

$\delta_{in}(q_9,GetEvents)=\varepsilon,\delta_{out}(q_9,GetEvents)=(q_8|q_5)^*$,

$\delta(q_{10},newspaper)=q_1.q_2.(q_4|q_6).(q_8^*|q_9^*)$。

可接受状态为 $A=\{q_{10}\}$。

然后，需要构造每一个转移函数右边定义的正则表达式所对应的 DFAs。这部分比较简单，限于篇幅这里不再叙述，详细内容请参考文献[11]。则 ASTA 机 B 定义了所有满足模式 s 的 AXML 文档集合。

不难看出，对于任意 AXML 模式均可构造等价的 ASTA 机。

3.2　AXML 模式重写判定算法

本文所提出的 AXML 模式重写判定算法其中心思想如下：首先根据 AXML 源模式 s 和目标模式 s'构造等价的 ASTA 机 B 和 B'，然后对 B'的转移函数通过附加函数节点的

输出类型进行扩展操作，再通过逐一检验 B 和 B' 转移函数之间的包含关系是否成立，来判定 AXML 模式重写。下面给出算法描述。

算法 SCHEMA_REWRITING(检验 AXML 模式重写)

输入：AXML 源模式 s 和目标模式 s′
输出：True or False
SCHEMA_REWRITING(s,s′)
begin
(a)根据模式 s 构造 ASTA 机 B = (Q,Σ,δ,A)
(b) 根据模式 s′构造 ASTA 机 B′ = (Q′,Σ,δ′,A′)
result:=True;
while(result) do
for (l∈Σ_L′) do
if δ(q,l) ≠ϕ and δ′(q′,l)≠ϕ then
Let M′:=δ′(q′,l);
for j=1,…,k do
for all edges e=(v,u) in M_j labeled with function name f_i do
extend M′ By attaching automaton M_{fi}, with its initial and final accepting states linked to v and u by ε edges, respectively;
if M⊆M′ then result:=True;
else result:=False;
return(result);
end

定理 算法 SCHEMA_REWRITING 是可终止的、正确的、完备的，复杂度为多项式时间。

证明：(1)算法终止性证明。算法首先构造与 AXML 模式 s 和 s' 等价的 ASTA 机 B 和 B'，可线性时间线终止；然后检验两个自动机对的包含关系来完成 AXML 模式重写的判定，该算法已知为可多项式时间终止的[11]。综上所述，算法是可终止的。

(2)算法正确性证明。假设 $B\subseteq B'$ 成立，意味着所有在 ASTA 机 B 上存在可接受运行的 AXML 文档，在 ASTA 机 B' 上亦存在可接受运行。由 ASTA 机的定义可知，ASTA 机 B 和 B' 分别定义了符合 AXML 模式 s 和 s' 的文档集合。考虑任意符合模式 s 的 AXML 文档 t，有 $t\in L(B)$ 成立，由假设可知 $t\in L(B')$ 成立，即文档 t 可重写为目标模式 s' 的文档实例，根据模式重写的定义，得知模式 s 可重写为目标模式 s'。

(3)算法完备性证明。完备性证明采用反证法。假设 $B\subseteq B'$ 不成立，但是模式 s 可重写为目标模式 s'。考虑任意符合模式 s 的 AXML 文档 t，由于 ASTA 机 B 等价于模式 s，则 t 在 B 上存在可接受运行，即有 $t\in L(B)$ 成立。根据假设 s 可重写为目标模式 s'，可知文档 t 已经是模式 s' 的文档实例或者存在 AXML 文档 t' 符合模式 s'，满足 t' 可由文档 t 生成，而 ASTA 机 B' 与模式 s' 等价，上述两种情况均说明文档 t 在 ASTA 机 B' 上存在可接受运行，即有 $t\in L(B')$ 成立，与假设矛盾。

(4)算法复杂度分析。算法首先根据 AXML 模式 s 和 s' 构造两个等价 ASTA 机，需要线性时间；接下来，包含两个 $|\Sigma|$ 次迭代的 for 循环，通过检验两转移函数的包含关系是否成立，该操作的运行时间已知为 $O(|Q||Q'||\Sigma|)$[11]。综上，算法 SCHEMA_REWRITING 的时间复杂度为 $O(|Q||Q'||\Sigma|^2)$。证毕。

4 实验分析

为研究 AXML 模式重写检验算法的有效性，本文完成了算法与文献[2]中提出的 AXML 模式重写算法的比较实验。实验通过改变 AXML 模式定义中函数元素的数目来测试算法的执行效率。模式中包含的规则定义类似于表达式 $r:=xyz(AB|OE)^*(AO|BE)^*(A|B)^*$，其中大写字母表示函数元素。实验分为两组：一组固定源模式大小，改变目标模式中函数元素的数目；一组是固定目标模式，改变源模式中函数元素的数目。实验目标是测试算法中 expand 方法与检验 ASTA 机中转移函数包含这两部分。本文采用 JAVA 语言实现了提出的模式检验算法，文献[2]的算法从 http://activexml.net 获得。实验环境为 PC 机 PIII1.2GHzCPU 256MB 内存，操作系统为 Fedora Core 6，实验结果如图 3 和图 4 所示。

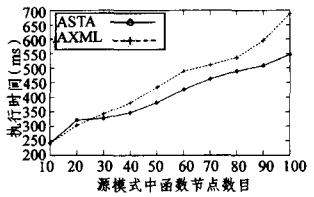

图 3 改变源模式中函数节点数目

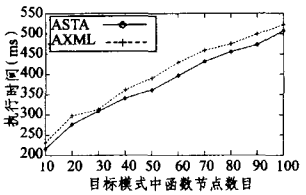

图 4 改变目标模式中函数节点数目

在增加源模式中函数节点数目的情况下，本文算法执行效率要高于 AXML 算法，这表明本文算法中的函数节点的扩展方法的执行时间低于文献[2](如图 3 所示)。这是因为本文算法并不记录转移函数自动机中用于判定调用序列的分叉节点，直接判定模式自动机的包含关系；在改变目标模式定义中函数节点数据的情况下，本文算法比 AXML 算法提高不是十分明显(如图 4 所示)。

结束语 本文基于树自动机理论提出了 AXML 模式重写判定算法，实验结果证明了算法的优越性。未来工作是对 ASTA 机的优化及如何利用 ASTA 机处理 AXML 查询等问题。

参 考 文 献

[1] Abiteboul S, Benjelloun O, Manolescu I, et al. Active XML: peer-to-peer data and Web services integration[C]// Proc. of the International Conference on Very Large DataBases. HongKong: Morgan Kaufmann, 2002: 1087-1090

[2] Milo T, Abiteboul S, Amann B, et al. Exchanging intensional XML data[C]// Proc. of the ACM SIGMOD International Conference on Management of Data. NewYork: ACM Press, 2003: 289-300

[3] Abiteboul S, Benjelloun O, Cautis B, et al. Lazy Query Evaluation for Active XML[C]// Proc. of the ACM SIGMOD International Conference on Management of Data. New York: ACM Press, 2004: 227-238

[4] Abiteboul S, Benjelloun O, Milo T. Positive active XML[C]// Proc. of the ACM Symposium on Principles of Database Systems. New York: ACM Press, 2004: 35-45

[5] Neven F. Automata, logic and XML[C]// Proc. of the 16th Intl Workshop Computer Science Logic. London: Springer, 2002: 2-26

(下转第 168 页)

合于描述不规则的并行[3]，将 OpenMP 对并行的描述能力由主要在循环级扩展到了循环级和任务级，这使得 OpenMP 能够适应更广泛的并行编程应用。已经有研究将事务存储扩展到了 OpenMP[4,5]。我们将本文提出的新语义扩展到 OpenMP API 3.0，设计并实现了相应的接口函数，下面是具体的扩展情况。

pragma omp transaction [clause[[,] clause]…]
structured-block

其中 clause 是 ordered，(open | closed) nested。ordered 要求执行事务的各线程以串行的顺序提交，如果没指明，默认进行未排序的串行事务。open nested 指明该事务是开放嵌套方式，close nested 指明该事务是封闭嵌套方式，默认情况下是 close nested。

为实现相应的语义，我们定义了读取开放嵌套事务提交结果的函数 void read_opennested_tran (varlist)。这个函数的输入是一个共享变量列表，结果分几种情况：如果没有开放嵌套事务更新了这些变量，那么保持当前事务最近见到的这些变量的值。即如果是第一次访问该变量，则从内存读取该变量。如果前面已经读取过了，那么就返回当前事务中该变量的值。如果有开放嵌套事务更新了这些变量或者部分变量，则更新对应变量的值，并且确定当前事务所在的最外层事务，与修改变量的开放嵌套事务所在的最外层事务一起提交。

OpenMP API 3.0 其他主要指导命令的扩展如图 3 所示，为了避免过于复杂的情况，我们不对这些指导命令进行事务的嵌套，这些指导命令一般作为外层事务。transfor 重用 for 的部分从句，某些从句作为扩展或者增加来描述相关循环体的事务特性，基本上对应于原来的含义，可以在其中使用事务指导命令。Section 和 task 指令命令的扩展与 for 类似。

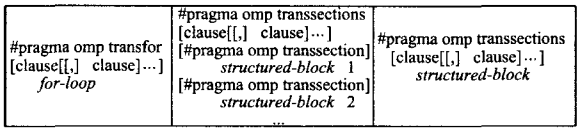

图 3 OpenMP 指导命令的事务存储扩展

现在考虑图 2 中的情况，以基于 OpenMP 的事务存储扩展来解决。如图 4 所示，线程 1 通过 task 指导命令嵌套了一个开放嵌套的事务，将 flagA 的值改为 true。线程 2 通过相应函数读取到了该值，并保证与线程 1 的事务同时推荐，以保持对共享变量修改的一致性，并从此刻开始重新计算对 flagA 变量的冲突检测，因此不会导致外层事务的冲突。线程 2 通过开放嵌套事务将 flagB 的值改为 true，同样线程 1 通过特殊函数读取该变量的值。两个外层事务完成了同步，分别完成了对 m 和 n 的更新。如果其中有外层一个事务与其他事务发生冲突而要作废，那么这两个外层事务都要作废。只有两个外层事务都能够成功提交时，它们的修改才生效，保持对共享变量修改的一致性。

```
flagA, flagB = false;
线程1                                   线程2
#pragma omp transtask                   #pragma omp transtask
 {                                       {
 #pragma omp transaction opennested      read_opennested_tran(flagA);
  {flagA = true;}                         while (!flagA)
 read_opennested_tran(flagB);             {read_opennested_tran(flagA);}
 while (!flagB)                          #pragma omp transaction opennested
  {read_opennested_tran(flagB);}          {flagB = true;}
 //update m                              //update n
 }                                       }
```

图 4 基于开放嵌套事务存储语义的事务同步

结束语 事务存储为程序员提供了一种高层的具有事务特征的并行执行模型和一种满足事务原子性(atomicity)、一致性(consistency)、隔离性(isolation)的共享存储访问机制。但目前事务存储的语义并不完善，事务间不能交换中间结果，不能实现锁的部分语义，某些情况下甚至将导致活锁。本文提出并实现了一种基于开放嵌套的事务存储的同步语义，并且将其扩展到了 OpenMP API 3.0，为事务间同步提供了一种解决方法，增强了事务存储的语义，解决了事务间不能交换中间结果的问题，增强了扩展事务存储后 OpenMP 的并行编程能力。在下一步的研究工作中，我们将进一步探讨在复杂的嵌套情况下，如何更有效地进行事务间的同步问题。

参 考 文 献

[1] 安虹，陈国良. 并行程序设计模型和语言[J]. 软件学报，2002，13(1)：118-124

[2] 彭林，张小强，谢伦国. 事务存储研究[C]// 中国计算机大会论文集. 2008

[3] Meadows L. OpenMP 3.0-A Preview of the Upcoming Standard [J]. Lecture Notes in Computer Science，2007，4782：4

[4] Baek W，Min C C，Trautmann M，et al. The OpenTM Transactional Application Programming Interface[C]// Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques. 2007：376-387

[5] Milovanovic M，et al. Transactional Memory and OpenMP[C]// Practical Programming Model for the Multi-Core Era，Proceedings. 2008，4935：37-53

[6] Blundell C，et al. Deconstructing Transactional Semantics：The Subtleties of Atomicity[C]// Workshop on Duplicating，Deconstructing，and Debunking (WDDD). 2005

[7] Ni Y，Menon V，Adl-Tabatabai A-R，et al. Open nesting in software transactional memory[J]// Proceedings of the 12th ACM SIGPLAN Symposium on PPOPP. 2007：68-78

[8] Herlihy M，Eliot J，Moss B. Transactional Memory：Architectural Supportor Lock-free Data Structures[C]// Proceedings of the 20th Annual International Symposium on Computer Architecture. 1993：289-300

[9] Shavit N，Touitou D. Software transactional memory[J]. Distributed Computing，1997，10(2)：99-116

(上接第 165 页)

[6] Neven F. Automata Theory for XML Researchers [J]. ACM SIGMOD Record，2002，31(3)：39-46

[7] Murata M，Lee Dongwon，Mani M，et al. Taxonomy of XML schema languages using formal language theory [J]. ACM Transactions of Internet Technology，2005，5(4)：660-704

[8] 高军，杨东青，唐世渭，等. 一种基于 DTD 的 XPath 优化方法[J]. 软件学报，2004，15(12)：1860-1868

[9] 高军，杨东青，唐世渭，等. 基于树自动机的 XPath 在 XML 数据流上的高效执行[J]. 软件学报，2005，16(2)：223-232

[10] Bruggemann-Klein A，Wood D. One-unambiguous regular languages[J]. Information and Computation，1998，142(2)：182-206

[11] Hopcroft J E，Motwani R，Ullman J D. Introduction to Automata Theory，Languages，and Computation[M]. Beijing：Tsinghua University Press，2002