2008 年 1 月 第 1 期
Vol. 29 No. 1 2008

DTD 可选的 XML 访问控制研究

曹益华[1], 张 昱[1,2]

[1](中国科学技术大学 计算机科学技术系, 安徽 合肥 230027)
[2](中国科学院 计算机科学重点实验室, 北京 100080)
E-mail: yuzhang@ustc.edu.cn

摘 要: 为解决XML 管理安全问题, 提出了DTD 可选的XML 访问控制系统OD-XACS(XML Access Control System with Optional DTD), 并给出了安全性分析. OD-XACS 支持访问控制规则中带有{//, *, [ ]}的复杂XPath 式. 有DTD 时, OD-XACS 利用XPath 式对DTD 的可满足性验证访问控制规则的有效性, 并对由规则中XPath 式构造的不确定有限自动机进行具体化, 消除了这些XPath 式中的冗余. 实验表明, 访问控制规则的验证和具体化可以极大地减轻XML 查询引擎的负担

关 键 词: DTD; XPath; 具体化; 访问控制; 自动机
中图分类号: TP311 **文献标识码**: A **文 章 编 号**: 1000-1220(2008)01-0073-07

XML Access Control with Optional DTD

CAO Yi-hua[1], ZHANG Yu[1,2]

[1](*Department of Computer Science & Technology, University of Science & Technology of China, Hefei* 230027, *China*)
[2](*Lab. of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing* 100080, *China*)

Abstract: To solve XML managerial security problem, an XML Access Control System with Optional DTD (OD-XACS) is proposed and its security is analyzed. OD-XACS supports complex XPath with {//, *, [ ]} in access control rules. With DTD, access control rules are validated according to XPath satisfiability. Moreover, XPath redundancy in access control rules is partially eliminated by NFA materialization. Experimental results show that validation and redundancy elimination of rules can relieve the burden on XML query engine.

Key words: DTD; XPath; materialization; access control; automata

1 引 言

XML (eXtensible Markup Language)的普遍应用使得用户对XML 数据的安全访问需求日益加强, 例如在Web 服务中要求服务器拒绝未授权用户对敏感信息的访问请求. XML 安全包括通信安全和管理安全[1], 本文只讨论管理安全. 一般采用访问控制技术作为XML 管理安全的解决方案.

XML 访问控制系统按预先设定的访问控制规则对用户访问条件进行权限检查, 输出权限许可的用户访问条件. 访问控制规则和用户访问条件一般使用XPath 表达式[2]来寻址XML 文档中的目标节点.

XML 访问控制的一般做法是一条访问控制规则直接作用到单个目标节点(可以是XML 文档节点[1,3,4]或DTD 等模式文档节点[5,6]), 再使用传播策略使其间接作用到目标节点的子孙节点等. 另一些研究[7,8,16]在访问控制规则中使用复杂XPath 式, 一条规则可以作用到多个目标节点. 其中[7]根据给定的DTD 文档计算规则中复杂XPath 式所匹配的目标模式节点, 但是没有给出该计算过程; QFilter[8]和[16]使用自动机进行访问条件的权限检查, 它们不利用模式文档计算规则匹配的目标节点, 因此可能输出在模式文档上无效的访问条件. 另外QFilter 虽然支持带{//, *, [ ]}的XPath 式(简称为$XP^{\{//,\ *,\ [\ ]\}}$), 但是不允许子孙轴(//)和谓词([ ])同时出现在规则中.

本文旨在设计实现一种DTD 可选的XML 访问控制系统OD-XACS (XML Access Control System with Optional DTD), 支持以$XP^{\{//,\ *,\ [\ ]\}}$为基础的访问控制规则和访问条件. 它既能在无DTD 时输出满足规则的访问条件, 也能在有DTD 时尽可能只输出满足DTD 约束的有效访问条件.

本文的主要贡献如下:

1) 有DTD 时, 给出访问控制规则验证和具体化算法, 该算法能够: 通过检验XPath 式对DTD 的可满足性, 排除无效访问控制规则; 消除访问控制规则中主XPath 式(见定义2)内的冗余路径, 删除各XPath 式内确定无效的谓词表达式.

2) 给出DTD 可选的访问条件权限检查算法, 它允许//和[ ]同时出现在访问控制规则中; 并给出安全性分析;

3) 将上述算法实现到OD-XACS 中, 实验表明有DTD 时OD-XACS 可以避免将大量的无效访问条件提交给XML 查询引擎, 从而大大减轻查询引擎的负担.

收稿日期: 2006-09-12 收修改稿日期: 2006-12-19 基金项目: 国家自然科学基金项目(60673126)资助; 中国科学院计算机科学重点实验室开放课题基金项目(SYSKF0502)资助. 作者简介: 曹益华, 男, 1982 年生, 硕士研究生, 研究方向为XML 安全计算; 张 昱, 女, 1972 年生, 博士, 副教授, 主要研究方向为XML 数据处理, 程序设计语言理论和实现技术.

论文的其余部分是这样组织的: 第 2 部分介绍后文用到的基本概念; 第 3 部分描述OD-XACS 框架; 第 4 部分给出基于DTD 的访问控制规则验证和具体化算法; 第 5 部分说明如何进行访问条件权限检查; 第 6 部分为实验结果及分析; 第 7 部分对全文进行总结

2 基本概念

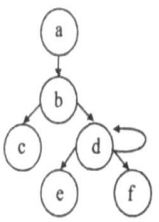
2.1 DTD

我们考虑DTD 的一个精简子集, 即DTD 中只包含元素声明. 基于该子集的算法可以很容易地扩展并应用到包含属性声明的DTD.

DTD 文档声明每个元素的内容模型(如图1(a)), '?, *, + '特别是' | '等操作符的存在增加了内容模型的复杂性. 在实际应用中, 内容模型有很多表示方法. 如: Xerces[12]使用二叉树保存内容模型, 进而对XML 文档进行有效性验证; [5]使用正规化DTD 以降低内容模型的复杂性, 在正规化DTD 中各元素的内容模型最多只出现一种操作符; [10]和[13]引入了DTD 图(如图1(b)). DTD 图简化了DTD, 并且不像正规化DTD 需要比较多的转换工作. 我们和[10]一样考虑DTD 图. 下面给出DTD 图的定义[10].

```
<! ELEMENT a(b,b)>
<! ELEMENT b(c?,d)>
<! ELEMENT c(#PCDATA)>
<! ELEMENT d(d|e+,f*)>
<! ELEMENT e(#PCDATA)>
<! ELEMENT f(#PCDATA)>
```

(a)



(b)

图1 DTD 文档(a)表示为DTD 图(b)

Fig. 1 Corresponding DTD graph (b) of DTD (a)

定义1. 给定DTD 文档τ, τ对应的DTD 图定义为一个有向图$G=(N, C)$, 其中每个节点$n \in N$ 对应于τ中元素, 每条弧$c=<n_1, n_2> \in C$, 表示τ中n_1 的元素声明中出现了n_2.

2.2 XPath

图2 给出本文所讨论的XP{//, *, [ ]}的形式化定义

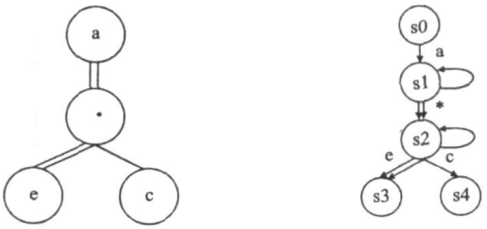
```
[1]P::= /E |//E
[2]E::= E/E |E//E |E[Q] |label |text() |* |@* |@label
[3]Q::= E |E OP Const |Q and Q |Q or Q |not(Q) |func(Q*)
[4]OP::= < |≤ |> |≥ |= |≠ |* |div |+ |-
```

图2 OD-XACS 支持的XPath 片段

Fig. 2 XPath fragment supported by OD-XACS

对上述XPath 式, 一般采用树模式(Tree Pattern, TP)[14]或自动机[8, 15]描述(如图3). 限于篇幅, 这里不详细描述树模式和自动机的构造过程, 只给出后文用到的相关概念.

定义2. 给定XPath 式p, 其对应树模式为T_p, 自动机为F_p. p 中去除所有谓词后的XPath 式称为p 的主XPath 式. T_p 中的结果节点是匹配主XPath 式的节点, 即主XPath 式的最后一个位置步到达的节点; 叶子节点是匹配谓词中的XPath 式的节点; 分支节点是主XPath 式对应的路径上具有的、分支到谓词中XPath 式的节点. F_p 中的相应状态称为结果状态、叶子状态、分支状态.



(a) (b)

图3 XPath 式/a//* [c]//e 的树模式表示(a)和自动机表示(b)

Fig. 3 Corresponding tree pattern (a) and automata (b) of XPath /a//* [c]//e

图3 为XPath 式/a//* [c]//e 对应的树模式(图3(a))和自动机(图3(b)). 树模式中圈表示节点, 其中e 为结果节点, c 为叶子节点, * 为分支节点. 自动机中圈表示状态, 其中s3 为结果状态, s4 为叶子状态, s2 为分支状态. 两图中, 孩子轴和子孙轴分别用单线和双线表示.

2.3 访问控制规则

一条访问控制规则(Access Control Rule, ACR)是一个三元组, 定义如图4.

```
[5]R::= (Name, Object, Action)
[6]Name::= QNAME
[7]Object::= P
[8]Action::= Action Action |read |update |insert |delete
```

图4 OD-XACS 支持的ACR

Fig. 4 ACR supported by OD-XACS

为叙述方便, 下文简称访问控制规则为规则, 并且只考虑read 操作, 因此忽略Action, 同时忽略Name. 这样本文的ACR 即简化为Object, 它定义为图2 中[1]描述的XPath 式.

2.4 强制执行

确保安全策略被正确执行的过程被称为强制执行(Enforcement)[3], 在本文中对应的过程是访问条件权限检查, 即根据规则检查用户访问条件输出满足规则限制的可访问条件.

本文中访问条件权限检查有如下两点限制:

2.4.1 不解析规则中谓词的内部结构

规则中的谓词是对其所属位置步节点测试进行进一步约束, 那么在用户访问条件对应位置步添加该谓词即可正确地强制执行该谓词约束, 无需解析规则中谓词的内部结构.

2.4.2 不解析访问条件中谓词的内部结构

因为规则限制的是对DTD 节点的访问, 对访问条件中的谓词进行权限检查也就是对访问条件谓词中的XPath 式进行权限检查, 这与检查访问条件中的主XPath式的访问权

限本质上是一样的, 而对访问条件中的谓词进行权限检查后如何构造可访问条件谓词以保持用户访问条件谓词的语义是我们的下一步工作.

这样, 访问条件权限检查的过程为: 首先只分别考虑访问条件和规则中的主XPath 式集合, 以这两个XPath 式集合在语义上的交集(详见第5 节)作为输出的可访问条件的主XPath 式集合; 然后在该集合中各XPath 式的对应位置步上分别添加访问条件XPath 式和规则XPath 式的谓词字符串, 得到最终的可访问条件集.

3　访问控制系统OD-XACS 框架

我们设计的XML 访问控制系统OD-XACS 主要任务是: 对于给定的访问条件集合、规则集合和可选的DTD 文档, 执行正确的访问条件权限检查, 输出可访问条件集合.

3.1　模块组成

OD-XACS 主要由五个模块组成, 见图5 中标号为1～5 的矩形框.

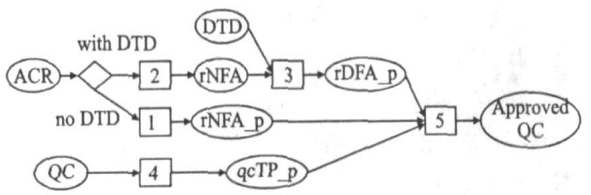


图5　OD-XACS 模块组成

Fig. 5　Overview of OD-XACS

· **模块1(rNFA-p 构造)**. 无DTD 时, 由ACR 中的所有主XPath 式构造一个规则NFA (Nondeterministic Finite Automata), 记为rNFA-p, 同时, 在各分支状态保存对应的谓词字符串;

· **模块2(rNFA 构造)**. 有DTD 时, 由ACR 中的所有XPath 式(包括主XPath 式和谓词中的XPath 式)构造一个规则NFA, 记为rNFA;

· **模块3(rDFA-p 构造)**. 有DTD 时, 构造DTD 图并输入到rNFA, 运行rNFA 得到对应于ACR 中所有主XPath 式的一个规则DFA (Deterministic Finite Automata), 记为rDFA-p, 同时, 在各分支状态保存对应的谓词字符串. 该过程包含去除无效规则和将规则中的* 或//按DTD 约束展开为确定路径的算法, 分别称为规则验证和具体化, 详见第4 节;

· **模块4(qcTP-p 构造)**. 由访问条件(Query Conditions, QC)中的所有主XPath 式构造一个树模式, 记为qcTP-p, 同时, 在各分支节点保存对应的谓词字符串;

· **模块5(访问条件权限检查)**. 将qcTP-p 作为输入运行rNFA-p 或rDFA-p, 进行访问条件权限检查, 得到可访问条件集(Approved QC).

模块1、2、4 中各自动机和树模式的构造过程类似, 我们集中在3.2 节讨论; 第4 节讨论rDFA-p 的构造, 包括规则验证和具体化; 第5 节讨论访问条件权限检查.

3.2　规则自动机和访问条件树模式构造

有DTD 时, rNFA 通过修改XML 查询引擎XSEQ[15]的部分特性来得到. 我们为规则中的XPath 式按共享前缀的方式增量式地构造rNFA, 其中的部分状态按定义2 被标记为结果状态、叶子状态或分支状态, 同时在这些状态上添加索引以加快rNFA 的运行: 在结果状态上添加LR 表, 保存匹配该状态的各个主XPath 式; 在叶子状态上添加LP 表, 保存匹配该状态的各个XPath 式及其所在的谓词; 在分支状态上添加LB 表, 保存该分支下的各个谓词[15].

由2.4 节限制1), 访问条件权限检查不解析规则中的谓词. 因此, 无DTD 时, 模块1 构造的自动机rNFA-p 与rNFA 的区别是, 前者只为ACR 中的主XPath 式构造相应的自动机状态. 这样rNFA-p 中没有谓词中的XPath 式对应的状态, 即没有叶子状态. 有DTD 时, 自动机rDFA-p 的构造详见4.2 节.

由2.4 节限制2), 本文不讨论访问条件中谓词的权限验证. 因此模块4 树模式qcTP-p 只为访问条件中的主XPath 式构造相应的节点. 其构造过程与rNFA-p 类似, 区别在于qcTP-p 是一个树状结构, 没有状态转换. 与自动机类似, qcTP-p 部分节点被标记为结果节点或分支节点, 并分别添加LR 表和LP 表作为索引.

4　规则验证和具体化

当存在DTD 文档时, 我们首先验证规则中XPath 式的有效性, 然后对有效规则进行具体化, 以消除XPath 式中//和* 的不确定性引起的冗余. 这一过程在OD-XACS 中体现为: 将DTD 图输入到rNFA 并运行, 得到rDFA-p.

4.1　规则验证

定义3. 给定DTD 文档τ和XPath 式p, 符合τ的所有XML 文档集合记为T, 对任意$t\in T$, p 在t 上计算得到匹配的XML 节点集[1]N_t, p 在τ上被称为:

1) 确定有效: 当$N_t\neq\varnothing$时;

2) 确定无效: 当$N_t=\varnothing$时;

3) 无法判断: 当N_t可能为空, 也可能不为空时.

我们将p 在τ上确定无效简称为无效, p 在τ上确定有效或无法判断合称为有效, 或称为可满足[9].

首先将DTD 表示为DTD 图, 然后在该图上通过深度优先遍历产生输入到rNFA 的SAD (Simple API for DTD)事件. 如果DTD 图中存在环, 我们将根据实际XML 文档的深度规定对环的展开深度, 从而消除环. 例如, 图6(见下页)为图1(b)对应的展开后的DTD 图并转化为SAD 事件, 图中d 上的自循环按深度为2 展开. 图6 中标注的SD、SE、EE、ED 分别是SAD 事件startDTDDocument、startElementDecl、endElementDecl、endDTDDocument 的简写, SE 和EE 后跟的

[1] 为叙述方便, 我们不讨论匹配结果为字符串的XPath 式.

参数a 表示元素名.

rNFA 的运行主要由SAD 事件处理规则定义. 在rNFA 运行过程中, 如果某结果状态 s 得到匹配, 并且LR(s)中某XPath 式(对应某条规则 r)的所有谓词也得到满足, 则 r 满足输入的DTD, 即 r 为有效规则.

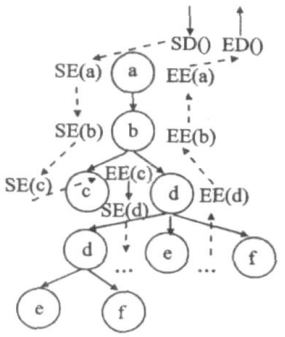


图6 DTD 图转化为SAD 事件

Fig. 6 Generating SAD events according to DTD graph

算法1 规则验证和具体化

规则验证: $s0$ 为初态, *stack* 为栈, 栈帧为rNFA 状态集

规则具体化: $d0$ 为初态, *dstack* 为栈, 栈帧为rDFA-p 状态

startDTDDocument()
(1) *Push*({$s0$}, *stack*);
(2) $ds0$= *newDState*({$s0$});
(3) *Push*($ds0$, *dstack*);

startElementDecl(a)
(1) *newsset*= *trans*(*stack*, *a*);
(2) *Push*(*newsset*, *stack*);
(3) *newrsset*= *getNFAResultStates*(*newsset*);
(4) *nds*= *newDState*(*newrsset*);
(5) *Push*(*nds*, *dstack*);
(6) for each *s* in *newsset*
(7) for each < *p*, *pred* > in LP(*s*)
(8) *evaluatePred*(*pred*, *p*);
(9) for each *p* in LR(*s*)
(10) *setRes*(*p*);

endElementDecl(a)
(1) *sset*= *Pop*(*stack*);
(2) *ds*= *Pop*(*dstack*);
(3) if (*ds*! = null)
(4) *prevds*= *Pop*(*dstack*);
(5) if (*prevds*= = null)
(6) *prevds*= *newDState*();
(7) add an edge from *prevds* to *ds* labeled *a*;
(8) *Push*(*prevds*, *dstack*);
(9) *copySatisfiedLB*(*ds*, *sset*);
(10) for each *s* in *dstack*
(11) for each *p* in LR(*s*)
(12) if (*getRes*(*p*) and all predicates of *p* satisfied)
(13) *setSatisfied*(*s*);
(14) for each *s* in *sset*
(15) for each *pred* in LB(*s*)
(16) *resetPred*(*pred*);

endDTDDocument()
(1) *deleteUnsatisfiedDStates*();
(2) foreach predicate *pred* in rule XPath set
(3) *deleteInvalidPredLogicExp*(*pred*);
(4) stop or wait another DTD document;

算法1 中包含两部分内容, 我们目前只讨论其中的规则验证部分. *trans*(*stack*, *a*)按照以下规则对 *stack* 的栈顶状态集进行状态转换: 当前状态的所有后继状态中由标记为/*a*、/*、//*a* 或//* 的转换边到达的状态都是匹配状态.

在rNFA 的运行中, 规则中谓词的验证主要通过SAD 事件处理中的谓词计算来完成. 在SE 事件中, 如果rNFA 当前转换到的某个状态 s 是叶子状态, 由索引LP(s)得到 s 关联的每个谓词中的XPath 式 p 及其所属谓词 *pred*, 对 *pred* 进行谓词计算((6)～(8)行). 由定义3, 此时称 p 在输入的DTD 上有效. 在EE 事件中, 如果rNFA 当前回溯到的状态 s 是分支状态, 由索引LB(s)得到 s 关联的每个谓词 *pred*, 将这些 *pred* 的谓词计算状态进行复位((14)～(16)行).

evaluatePred(*pred*, *p*)自底向上计算整个谓词表达式 *pred* 的有效性. 类似定义3, 我们定义谓词中一般表达式 e 的有效性: 给定DTD 文档 τ 和表达式 e, 如果 e 在 τ 上计算为永假, 则 e 在 τ 上无效; 否则 e 在 τ 上有效.

例1 规则XPath 集合为{/a//* [c]//e, //d, /a//* [c]/e, /a/b[3]/* },DTD 同图1(a). 则验证后规则XPath 集合为{/a//* [c]//e, //d}. 其中/a//* [c]/e 无效是因为谓词[c]和主XPath 式不能同时满足, /a/b[3]/* 无效是因为索引谓词[3]无效.

4.2 规则具体化

定义4 给定规则 r= (*name*, *p*, *action*), 如果 p 的主XPath 式路径中不包含//或*, 称 p 为简单路径XPath 式, 称相应的 r 为简单路径规则.

由于XPath 式中//和* 的不确定性, 有效规则相对于简单路径规则可能存在不满足DTD 的冗余路径.

我们的做法是将有效规则展开为简单路径有效规则集合, 并构造相应rDFA-p, 称为rNFA 的具体化. 一种简单的rNFA 具体化算法可分为两个阶段:

1) 由规则验证并具体化得到简单路径有效规则集合;

2) 由简单路径有效规则构造DFA.

该算法主要缺点是需要临时保存简单路径有效规则.

为避免临时保存规则, 我们在规则验证的同时构造rDFA-p, 见算法1 规则具体化部分. 主要包括三个处理步骤:

1) 一旦rNFA 运行到达结果状态(SE 事件), 即构造相应的rDFA-p 结果状态((3)～(5)行), 并在回溯时(EE 事件)构造出从rDFA-p 结果状态到根状态的所有rDFA-p 状态, 它对应于一条简单路径XPath 式((2)～(8)行);

2) 在rNFA 回溯到分支状态时(EE 事件), 如果顶层谓词得到满足, 则在rDFA-p 对应分支状态添加该谓词引用

(第(9)行);

3) rNFA 运行完毕时(ED 事件), 删除不满足所有谓词约束的 rDFA - p 结果状态及相应转换边(第(1)行), 删除所有无效的谓词逻辑表达式((2)～(3)行).

方法 *deleteInvalidPredLogicExp*(*pred*) 自顶向下遍历谓词 *pred* 中的逻辑表达式, 根据谓词计算结果删除 *pred* 中的永假式, 我们称之为对谓词中无效逻辑表达式的逻辑优化. 与此接近的工作是[11], 该文讨论了无 DTD 约束下 XPath 式的逻辑优化.

例2 规则 XPath 集合为{/a//* [c]//e, //d}, DTD 同图1(a). 则具体化后的 rDFA - p 如图7 所示.

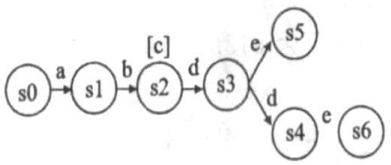


图7　例2 中的 DFA

Fig. 7　DFA in example 2

5　访问条件权限检查

访问条件权限检查的主要任务是: 将 qcTP - p 输入到 rNFA - p (或 rDFA - p) 并运行, 得到可访问条件集 QC_A.

如图8(对应 XPath 式为图3(a)), 我们通过深度优先遍历将 qcTP - p 转化为 SAP(Simple API for XPath) 事件, 图中标注的 SX、SN、EN、EX 分别是 SAP 事件 startXPath、startNodeTest、endNodeTest、endXPath 的简写, SN 和 EN 后跟的参数 n 表示当前树模式节点, 包含当前位置步的轴、节点测试以及该节点保存的索引信息(图中只标出轴和节点测试, 未反映索引信息).

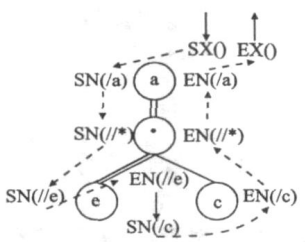


图8　树模式转化为 SAP 事件

Fig. 8　Generating SAP events according to tree pattern

rNFA - p (或 rDFA - p) 的运行主要由 SAP 事件处理规则定义.

算法2 访问条件权限检查

*s*0 为初态, *xstack* 为栈, 栈帧为 rNFA - p 运行时状态集(或 rDFA - p 状态)

startXPath()

(1) *Push*({*s*0}, *xstack*);

(2) QC_A. *clear*();

startNodeTest(**n**)

(1) *sset*= *xtrans*(*xstack*, *n*);

(2) for each *pq* in LR'(*n*)

(3)　for each *pr* in LR(each *s* in *sset*)

(4)　　*pa*= *newLocationStepList*(*s*);

(5)　　*palist*= *eliminatePredConflict*(*pa*);

(6)　　QC_A. *add*(*palist*);

endNodeTest(**n**)

(1) *sset*= *Pop*(*xstack*);

endXPath()

(1) stop or wait another tree pattern;

算法2 中 *xtrans* 按照表1 的状态转换规则对 rNFA - p (或 rDFA - p) 进行状态转换.

表1　xtrans 状态转换规则

Table 1　State transitions in xtrans

当前节点 n 对应的轴和节点测试	rNFA - p (或 rDFA - p) 状态转换规则
/*a*	当前状态的所有后继状态中由标记为/*a*、/*、//*a* 或//* 的转换边到达的状态都是匹配状态
/*	当前状态的所有后继状态都是匹配状态
//*a*	当前状态的所有子孙状态中由标记为/*a*、/*、//*a* 或//* 的转换边到达的状态都是匹配状态
//*	当前状态的所有子孙状态都是匹配状态

xtrans 同时为每个运行时状态 *s* 记录以下信息:

· 父状态 s_p

· 对应的 qcTP - p 节点 *n*

· 由 s_p 转换到 *s* 时 qcTP - p 或 rNFA - p (或 rDFA - p) 当前转换弧中更为具体的轴和节点测试 *pathSeg*. 其中, 孩子轴/比子孙轴//更具体, 节点测试/*a* 比/* 更具体.

newLocationStepList(*s*) 使用这些信息从结果状态 *s* 构造出可访问条件(由运行时状态的父子链连接), 此时可访问条件每个位置步的谓词为对应状态 LB 索引中属于 *pr* 的谓词和对应节点 LB 索引中属于 *pq* 的谓词, 并且规则 *pr* 的谓词约束在前.

eliminatePredConflict(*pa*) 是在 *pa* 中消除规则谓词的语义冲突, 因为规则主 XPath 式中的分支节点可能匹配 *pa* 中多个节点. 例如, 若 *R* = {/a//* [c]//e}, *pa*= /a//b/d/e, 则 *pa* 中的//b 和/d 都是匹配规则/a//* [c]//e 中分支节点//* 的位置步, 从而谓词[c]同时出现在 *pa* 的位置步//b 和/d 中, 这和规则本身的语义不一致, 消除冲突后应该形成两条符合规则的可访问条件, 即/a//b[c]/d/e 和/a//b/d[c]/e. 通过对 *pa* 复制副本并消除谓词冲突, *eliminatePredConflict* 确保在 *palist* 的每个可访问条件中每个规则谓词最多出现一次, 并且 *pa* 对应的所有可能的可访问条件都包含在 *palist* 中.

定理1 给出了算法2 处理结果的安全性分析.

定理1. 利用算法1 得到的可访问条件集是正确的, 即 $QC_A = QC \cap R$.

证明: 由QFilter 的工作[8], 我们只需要证明规则中XPath 式同时出现//和[ ]的情况

记$R = R' \cap PRED$, 其中R'为R 中所有主XPath 式, $PRED$ 为R 中的所有谓词约束; $QC_A = QC_A' \cap PRED_A$, 其中QC_A'为QC_A 中所有主XPath 式以及所有访问条件谓词约束, $PRED_A$ 为QC_A 中的规则谓词约束. 由QFilter 定理3 可知$QC_A' = Q \cap R'$.

由于nNFA-p(或nDFA-p)中的状态与QC_A 中的节点(对应到nNFA-p 或nDFA-p 的运行时状态)是一对多的关系, 显然, $PRED_A \subseteq PRED$.

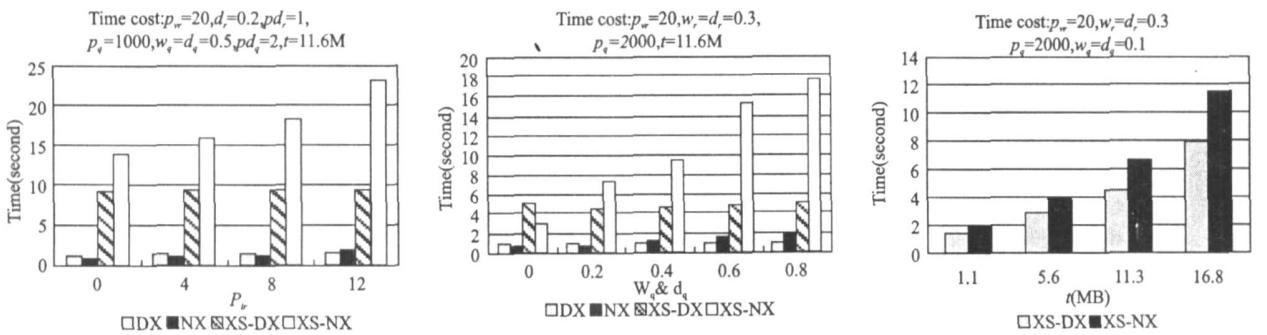
我们推导得$QC \cap R = QC \cap R' \cap PRED = QC_A' \cap PRED = [QC_A' \cap PRED_A] \cup [QC_A' \cap (PRED - PRED_A)] = QC_A \cup [QC_A' \cap (PRED - PRED_A)]$, 要证明结论$QC_A = QC \cap R$, 只需要证明$QC_A' \cap (PRED - PRED_A) = \varnothing$, $\varnothing$为空集

算法1 的*eliminatePredConflict* 确保$PRED_A$ 为$PRED$ 在QC_A'上应用的所有可能约束, 也就是说, 对谓词约束$PRED - PRED_A$, 在QC_A'中没有可施加约束的位置步, 则QC_A'经过$PRED - PRED_A$ 过滤后总是得到空的节点集合. 因此$QC_A' \cap (PRED - PRED_A) = \varnothing$.

例3 规则集R 为{/a//* [c]//e, //d}, 访问条件集QC 为{/a/b[. //d= "sth"]//e}. 以nNFA-p 为例, 运行得到pa_1: /a/b[c][. //d= "sth"]//e, pa_2: /a/b[c][. //d= "sth"]//*[c]//e, pa_1 没有谓词冲突, pa_2 消除谓词冲突后得到$palist_2$: {/a/b[c][. //d= "sth"]//* //e, /a/b[. //d= "sth"]//*[c]//e}. 从而最终的可访问条件集QC_A 为{/a/b[c][. //d= "sth"]//e, /a/b[c][. //d= "sth"]//* //e, /a/b[. //d= "sth"]//*[c]//e}.

6 实 验

我们用Java 实现了OD-XACS(分别记使用DTD 验证和不使用DTD 验证为DX 和NX). 使用查询引擎XSIEQ 执行DX 和NX 两者输出的访问条件, 分别记为XS-DX 和XS-NX. 实验平台为Windows XP 操作系统, CPU 为Athlon XP 1800+ 1.5GHz, 内存为512M DDR. 测试用的DTD 和XML 文档由XMark[17]得到, XPath 式(包括规则XPath 式和访问条件XPath 式)由YFilter[18]的XPath 生成工具根据DTD 生成. 实验考核的因素包括XML 文档大小t、访问条件XPath 式的个数p_q、有效规则XPath 式的个数p_{vr}、无效规则XPath 式的个数p_{ir}、单个规则XPath 式所含的谓词数pd_r、单个访问条件XPath 式所含的谓词数pd_q、规则XPath 式中通配的概率w_r 和子孙轴的概率d_r、访问条件XPath 式中通配的概率w_q 和子孙轴的概率d_q 等. 图9 为部分实验结果



(a) (b) (c)

图9 OD-XACS 中DX 和NX 算法时间比较

Fig. 9 The time cost of OD-XACS

(a)、(b)两图分别考察p_{ir}或w_q、d_q 对DX 和NX 验证时间(Check Time, CT)及相应查询时间(Query Time, QT)的影响. 显然, 增加p_{ir}或w_q、d_q 时, CT (NX)比CT (DX)增加的快, 甚至超过CT (DX); QT (XS-DX)基本不受影响, 而QT (XS-NX)迅速增加. 这是因为NX 忙于用无效规则或存在冗余的规则处理访问条件, 并可能输出无效的访问条件.

(c)图考察t 对QT (XS-DX)和QT (XS-NX)的影响. 我们可以看出QT (XS-NX)一直大于QT (XS-DX), 原因同上.

7 结论与后续工作

本文提出的OD-XACS 系统: 在无DTD 约束时支持了比QFilter 更为复杂的规则集合; 通过基于DTD 的规则验证和具体化, 排除了大部分无效规则, 并对有效规则进行了优化. 实验证明我们的方法是有效的.

我们下一步的工作包括两个方面, 一是对访问条件谓词中的XPath 式进行权限检查; 二是对本文逻辑优化的补充, 我们考虑对确定有效XPath 式进行判断, 由此可以优化确定有效的谓词逻辑表达式.

References:

[1] Chung-Hwan Lim, Seog Park, Sang H Son. Access control of XML documents considering update operations[C]. In: Proceedings of the ACM Workshop on XML Security, Fairfax, VA, 2003: 49-59.

[2] James Clark, Steve DeRose. XPath version 1.0. W3C recommendation[EB/OL]. http://www.w3.org/TR/xpath, 1999.

[3] Cho SungRan, Srivastava Divesh. Secure evaluation of XML queries[J]. Dissertation Abstracts International, 2003, 64(05)

B: 2261.

[4] Yu T, Srivastava D, Lakshmanan L, et al. Compressed accessibility map: efficient access control for XML [C]. In: Proceedings of the 28th VLDB Conference, Hong Kong, 2002.

[5] Fan W, Chan C Y, Garofalakis M. Secure XML querying with security views [C]. In: Proceedings of ACM SIGMOD Intl' Conference, Paris, France, 2004: 587-598.

[6] Xinwen Zhang, Jaehong Park, Ravi Sandhu. Schema based XML security: RBAC approach [C]. 17th IFIP Working Conference on Data and Application Security, Estes Park, Colorado, USA, 2003.

[7] Ernesto Damiani, Sabrina De Capitani di Vimercati, Stefano Paraboschi, et al. A fine-grained access control system for XML documents [J]. ACM Transactions on Information and System Security (TISSEC), 2002, 5(2): 169-202.

[8] Bo Luo, Dongwon Lee, Wang-Chien Lee, et al. QFilter: fine-grained run-time XML access control via NFA-based query rewriting [C]. In: 13th ACM Int'l Conf. on Information and Knowledge Management (CIKM), Washington DC, USA, 2004.

[9] Michael Benedikt, Wenfei Fan, Floris Geerts. XPath satisfiability in the presence of DTDs [C]. In: Proceedings of the twenty-fourth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2005: 25-36.

[10] Stefan Böttcher, Rita Steinmetz. A DTD graph based XPath query subsumption Test [C]. XML Database Symposium (XSym 2003) at VLDB, Berlin, Germany, 2003.

[11] Pierre Genevès, Jean-Yves Vion-Dury. Logic-based XPath optimization [C]. In: Proceedings of the 2004 ACM Symposium on Document Engineering, 2004: 211-219.

[12] Xerces2 Java Parser 2.6.0 [EB/OL]. http://xerces.apache.org/xerces2-j/.

[13] Stefan Böttcher. Testing intersection of XPath expressions under DTDs [C]. The 8th International Database Engineering & Applications Symposiu, Coimbra, Portugal, 2004.

[14] Gerome Miklau, Dan Suciu. Containment and equivalence for an XPath fragment [C]. In: Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, 2002: 65-76.

[15] Zhang Yu, Wu Nian. XSIEQ-an XML stream query system with immediate evaluation [J]. Mini-Micro Systems, 2006, 27(8): 1514-1518.

[16] Makoto Murata, Akihiko Tozawa, Michiharu Kudo, et al. XML access control using static analysis [C]. In: Proceedings of the 10th ACM Conference on Computer and Communications Security, Washington D.C., USA, 2003: 73-84.

[17] Albrecht Schmidt, Florian Waas, et al. XMark: a benchmark for XML data management [C]. In: Proceedings of the 28th VLDB Conference, Hong Kong, China, 2002.

[18] YFilter 1.0 Code Release [EB/OL]. http://yfilter.cs.berkeley.edu/code-release.htm, April 2005.

附中文参考文献:

[15] 张　昱, 吴　年. XSIEQ—一种立即计算的 XML 流查询系统 [J]. 小型微型计算机系统, 2006, 27(8): 1514-1518.