一种基于内容模型图的 XML Schema Definition 的提取方法

宁 静[1,2] 刘 杰[1,3] 叶 丹[1]

(中国科学院软件研究所软件工程技术中心 北京 100190)[1]

(中国科学院研究生院 北京 100190)[2] (中国科学技术大学计算机科学与技术系 合肥 230026)[3]

摘 要 使用 XML Schema 能够对 XML 文档进行有效性验证以及对查询、转换等操作进行优化,但是实际应用中大量 XML 文档缺失关联的 Schema。提出了一种根据 XML 文档自动提取对应的 XML Schema Definition (XSD)的方法 XSDInfer,先根据合并规则将 XML 解析过程中分析得到的模式信息记录在内容模型图中,再根据生成规则将其转化为内容模型表达式,进而得到 XSD。XSDInfer 能够在内存消耗比较少的情况下快速地处理超大规模、深度嵌套的 XML 文档,同时支持 XSD 的上下文相关的内容模型,提取出的 XSD 也具有更好的可读性。实验表明,与同类方法相比,XSDInfer 具有更好的可扩展性和表达能力。

关键词 XML,XML Schema Definition,模式提取,内容模型

中图法分类号 TP311 **文献标识码** A

Novel Approach for Extracting XML Schema Definition Based on Content Model Graph

NING Jing[1,2] LIU Jie[1,3] YE Dan[1]

(Technology Center of Software Engineering, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China)[1]

(Graduate University, Chinese Academy of Sciences, Beijing 100190, China)[2]

(Dept. of Computer Science and Technology, University of Science & Technology of China, Hefei 230026, China)[3]

Abstract Although XML Schema can be used to perform validation, querying and transformation on XML documents, a lot of XML documents in real applications have no XML Schema defined. This paper presented an approach, XSDInfer, to extract XML Schema Definition (XSD) from XML documents automatically. Firstly, schema information harvested from XML parsing was merged into the Content Model Graphs by applying rules. Then the graphs were transformed to content model expressions to generate the XSD. XSDInfer can scale to very large and deep recursive XML documents. It supports the context-sensitive content model, and the generated XSD is more human-readable. Experiments show that XSDInfer achieves better performance both in scalability and expressiveness in contrast to the previous techniques.

Keywords XML, XML schema definition, Schema extraction, Content model

1 背景介绍

XML 作为网络环境下重要的数据交换和数据存储方式,已经被广泛应用于电子商务、企业信息交换/集成、Web 服务等领域。XML Schema 是用来描述 XML 文档的结构和内容的语言,可以将 XML 文档中隐含的结构和类型信息明确地表述出来。通过为 XML 文档指定 XML Schema,不仅可以验证 XML 的有效性,而且能够对 XML 的查询、转换、集成等操作进行优化和自动化处理。此外,一些 XML 数据绑定工具,如 JAXB[1],也要求用户为 XML 文档提供 XML Schema。虽然使用 XML Schema 具有重要意义,但是实际上从 Web 上搜集到的 XML 文档中,约有一半并未附带 XML Schema[2]。因此,研究如何根据 XML 文档自动提取相应的 XML Schema 的方法是很有必要的。

文档类型定义(Document Type Definitions, DTD)和 W3C XML 模式定义(W3C XML Schema Definition, XSD)是 W3C 相继推出的 XML Schema 规范,是主流的 XML Schema 语言。近年来,XSD 由于其更丰富的表达能力已经逐渐取代 DTD 而被广泛应用。与 DTD 相比,XSD 不仅提供了更加丰富的预定义数据类型并支持命名空间,而且其元素的内容模型(Content Model)也具有更强的表达能力,体现在如下两个方面:

(1)内容模型的上下文相关性。在 DTD 中,元素的内容模型只依赖于元素的名字。而在 XSD 中,元素的内容模型还依赖于该元素所在的上下文[2]。例如,对于图 1 所示的 XML 片段,元素 name 的内容模型与其所在的上下文有关:若元素

到稿日期:2009-07-15 返修日期:2009-10-12 本文受国家 863 高技术研究发展计划(2007AA01Z149,2007AA04Z148)资助。

宁 静(1984—),女,硕士生,主要研究方向为 XML 数据集成、网络分布式计算,E-mail:ningjing07@otcaix.iscas.ac.cn;刘 杰(1982—),男,博士生,主要研究方向为软件工程、数据集成、数据质量;叶 丹(1971—),博士,高级工程师,CCF 高级会员,主要研究方向为分布式计算、中间件技术、企业应用集成。

name是元素department的子元素，则name用于表示部门的名称，是字符串类型；若name是manager或employee的子元素，则name用于表示人名，是复杂类型，可以包含子元素given和family。若使用DTD进行定义，则元素name只对应一种内容模型，其DTD片段为(! ELEMENT name (#PADATA | (given,family)))，可见它是无法区分上述差异的。而XSD则可以根据元素name所在的上下文定义两种不同的内容模型，如图2所示。

```
<department>
  <name>Marketing</name>
  <manager> <name>
      <given>Bob</given>
      <family>Lee</family>
    </name>
    <phone>8545622</ phone >
  </manager>
  <employee><name>
      <given>John</given>
      <family>White</family>
    </name>
    < phone >324568</ phone >
  </employee>
  <employee><name>
      <given>Kate</given>
      <family>King</family>
    </name>
    < phone >3658974</ phone >
  </employee>
</department>
```

图1 XML片段

```
<xs:element name="department">
 <xs:complexType><xs:sequence>
 <xs:element name="name"
type="xs:string"/>
 <xs:element name="manager"
type="personType"/>
 <xs:element name="employee"
type="personType" maxOccurs="unbounded"/>
 </xs:sequence>
 </xs:complexType>
 </xs:element>
 <xs:complexType name="personType">
 <xs:sequence>
  <xs:element name="name">
  <xs:complexType>
  <xs:sequence>
    <xs:element name="given"
type="xs:string"/>
    <xs:element name="family"
type="xs:string"/>
  </xs:sequence>
  </xs:complexType>
  </xs:element>
  <xs:element name="phone" type="xs:string"/>
 </xs:sequence>
 </xs:complexType>
```

图2 XSD片段

(2)对出现次数(Numerical Occurrence)进行精确限定。DTD的内容模型只使用"*","+","?"来表达出现"0或多次"、"1或多次"、"0或1次"，而XSD则能够通过"maxOccur"和"minOccur"精确限定最多和最少出现次数。

目前大量关于XML Schema提取方面的研究仍然集中在DTD上，对XSD的研究相对较少。现有的XSD的研究或工具[3-6]，大多未考虑内容模型的上下文相关性，因此得到的XSD在表达能力上等同于DTD，只有文献[2]进行了这方面的研究，但是它得到的XSD也不支持对元素出现次数的精确限定。

在文献[2,3]的基础上，本文提出了一种XSD的提取方法XSDInfer。我们提出了一种称为"内容模型图(Content Model Graph,CMG)"的数据模型，用于记录在XML解析过程中分析出的模式信息，并给出了构造CMG的算法eCMG-Merge和根据CMG生成内容模型的算法eGenerate。XSDInfer同时支持前述XSD的表达能力(1)和(2)。实验表明，与同类方法相比，XSDInfer具有良好的可扩展性，对于超大规模的XML文档(1GB)，XSDInfer占用的内存仅为文献[3]的1/3，并且XSDInfer具备处理同类方法难以处理的深度嵌套的XML文档的能力。此外，XSDInfer生成的结果在可读性方面也优于同类算法。

本文第2节叙述相关工作；第3节给出有关定义；第4节详细介绍XSDInfer算法；第5节通过实验给出XSDInfer与相关方法的比较；最后总结全文。

2 相关工作

DTD和XSD是两种主流的XML Schema语言，因此对XML Schema提取方面的研究也主要针对这两者。

DTD提取过程中的关键问题是如何得到元素的内容模型。文献[7]先为元素生成若干正则表达式，然后使用factoring算法提取公共子表达式，最后使用Minimum Description Length (MDL)规则选择出最佳表达式作为该元素的内容模型。文献[8]将元素的内容模型限制为每个子元素只能在内容模型中出现一次，使用一些启发式规则(Folding Rules和Relaxed Transformation Rules)生成正则表达式，然后使用文献[7]的factoring算法得到元素的内容模型。类似地，文献[9]也将内容模型限制为Single Occurrence Regular Expression (SORE)，先为每个元素构造一个有限自动机，然后使用算法*i*DTD将有限自动机转化为正则表达式，从而得到该元素的内容模型。文献[9]还提供了一个针对特殊的内容模型(Chain Regular Expression, CHARE)的模式提取算法(CRX)。

对于XSD提取的研究大多利用了DTD的内容模型生成方法。一些具有XSD提取功能的软件或工具，如Trang[6]，Atoval XMLSpy[4]，Stylus Studio[5]，生成的XSD在第1节所述的表达能力(1)上都是等同于DTD的。文献[2]给出了一个推理XSD类型的算法*i*XSD，*i*XSD使用文献[9]的*i*SOA[1]生成元素内容模型，并考虑到了XSD与DTD在表达能力(1)上的不同，但是得到的XSD在表达能力(2)方面与DTD是相同的。XStruct[3]将文献[8]的算法扩展为能够从多个XML文档中提取XSD，并增加了对数据类型和属性的分析。对于一个元素，XStruct先使用文献[8]的启发式规则为该元素的每一次出现生成一个内容模型实例，再基于文献[7]的factoring算法对所有的内容模型实例进行合并，最终得到该元素的内容模型。这种方法的缺点主要有如下两点：(i)未考虑内容模型的上下文相关性，得到的XSD在表达能力(1)上与DTD是相同的。(ii)只有当所有的内容模型实例都得到后才能使用factoring算法进行合并，这意味着在XML解析的过程中必须将所有不同的内容模型实例都存储起来，在XML结构不规则的情况下，这将占用大量的内存。此外，factoring算法的频繁、深度递归调用及相关数据结构也消耗了大量的内存，我们将在实验部分给出详细数据。文献[3]在总结部分指出了factoring算法本身的复杂性以及由factoring算法导致的内存消耗问题，并指出寻找一种"on the fly factoring"的算法是值得探索的。

3 定义

3.1 上下文相关元素

如第1节所述，XSD的内容模型具有上下文相关性。一般地，用"元素结点"来指代元素名在XML文档中的一次出现。事实上，XSD的内容模型是由当前元素结点到根结点的路径来唯一确定的[10]，因此可以通过维护路径信息来维护内容模型的上下文相关性。下面提到的内容模型都具有上下文相关性。术语"上下文相关元素"包含两层含义：①元素名；②该元素到根元素的路径。称两个元素结点是相同的上下文相

[1] 在文献[9]中称为*i*DTD。

关元素，当且仅当它们的元素名和到根的路径都相同。这样，XSD的提取过程就可以抽象成①识别出所有上下文相关元素和②为每个上下文相关元素生成内容模型的过程。

3.2 内容模型实例

XML Schema用内容模型(Content Model)来描述一个元素的内容和结构，内容模型可以被抽象为正则表达式。我们可以通过为XML文档中的每个元素定义内容模型来描述整个XML文档的结构。W3C规范要求XSD的内容模型必须符合Unique Particle Attribute (UPA)原则，即要求内容模型是确定性(deterministic)的正则表达式[11]。确定性正则表达式可以通俗地理解为：对于句子中的任意字符，无需向前看就可以确定使用正则表达式中的哪个字符进行匹配。例如有正则表达式$(ab?)^* a$和句子aaa，如果不向前看，就无法确定该句子中的第一个a应该使用正则表达式中哪个a进行匹配，所以$(ab?)^* a$不是确定性正则表达式。基于UPA原则，很多研究[2,3,8,9]进一步将内容模型限制为每个字符只出现一次的单次出现正则表达式(Single Occurrence Regular Expression, SORE)。例如，$ab?\ c^*$是SORE，而$ab^*\ a$不是SORE，因为a出现了两次。容易看出，SORE是确定性正则表达式。事实上，实际使用的XML Schema中，有99%的正则表达式是SORE[9]。为了描述XSD的内容模型，将SORE扩展为带有次数标记的单次出现正则表达式(Extended Single Occurrence Regular Expression, ESORE)，形如$(ab^{\langle 1,2\rangle} c)^{\langle 1,3\rangle}$。在本文中，将元素的内容模型限制为ESORE。

一个元素可能在XML文档中出现多次。本文将根据一个上下文相关元素某次出现时的子元素序列提取得到的正则表达式称为该上下文相关元素的一个内容模型实例。内容模型实例是对该上下文相关元素某次出现时的内容和结构的描述。基于文献[3]中的定义，进一步将内容模型实例定义为

$$E:=[T_1\cdots T_k]^{\langle \min,\max\rangle}$$

式中，T_n称为Term，是若干字符的序列(Sequence)或选择(Choice)，定义如下

1) AndTerm: $T_n:=(s_{n1}^{opt}\cdots s_{nj}^{opt})^{\langle \min,\max\rangle}$

2) OrTerm: $T_n:=(s_{n1}^{opt}+\cdots+s_{nj}^{opt})^{\langle \min,\max\rangle}$

其中，$\min\in\{0,1\}$，$\max\geqslant 1$，字符s_{bn}代表一个XML元素名，$opt\in\{true,false\}$，若$opt=false$，则该字符在Term中必须出现。可见，一个Term是对一组XML元素的描述，这组XML元素可以相继出现(AndTerm)或互斥出现(OrTerm)至少min次，至多max次。当min和max均为1时，可以省略标注。相应地，内容模型实例E是若干Term构成的序列，该Term序列可以重复出现min到max次。去除min，max和opt信息的Term称作原生Term，记为P_String，例如对于$T_n:=(s_{n1}^{opt}\cdots s_{nj}^{opt})^{\langle \min,\max\rangle}$，$P_String(T_n)=s_{n1}\cdots s_{nj}$。为了便于后面的叙述，将一个Term T_n中所包含的字符的集合定义为

$$\sigma(T_n):=\{s_{n1},s_{n2},\cdots,s_{nj}\}$$

对于一个内容模型实例E，要求$\sigma(T_p)\cap\sigma(T_q)=\varnothing$，$p,q\in\{1,2,\cdots,k\}$，$p\neq q$。即$E$中任意字符只出现一次，可知$E$是ESORE。

4 XSDInfer

针对第2节提到的XSD研究中存在的问题，本文提出了XSDInfer。下面将在4.1节—4.4节中介绍XSDInfer使用的内容模型存储模型——内容模型图(Content Model Graph, CMG)、XSDInfer算法描述以及XSDInfer使用的合并算法eCMGMerge和生成内容模型表达式的算法eGenerate。由于属性和数据类型的推理相对更为简单，因此只给出内容模型的提取方法。

4.1 内容模型的存储结构

我们引入结构模型(Structure Model, SM)来记录XML文档(集)所有上下文相关元素的信息。SM是一个哈希表，键(key)为元素到根元素的路径。从前述可知，该路径维护了元素的上下文相关性。哈希表的值(value)是上下文相关元素对应的内容模型图(CMG)。XSDInfer为XML中的每个上下文相关元素维护一个CMG，以此来保存该元素可能出现的内容和结构。CMG可以被转化为ESORE(具体的转化算法将在4.4节给出)，从而得到该元素的内容模型。内容模型图CMG是一个有向图，定义如下

$$CMG:=(V,L,s,f,\min,\max,\varphi)$$

式中，V是结点的集合，每个结点表示一个Term，并且要求对于任意两个Term T_i和T_j，$\sigma(T_i)\cap\sigma(T_j)=\varnothing$。$L$是有向边的集合，表示Term之间的位置关系。$s$和$f$是标记结点，$\sigma(s)=\varnothing$，$\sigma(f)=\varnothing$，$s$是CMG中唯一的入度为0的结点(起点)，$f$是CMG中唯一的出度为0的结点(终点)。min和max分别记录内容模型的最小值和最大值。φ是字符到该字符所在Term的映射。例如，对于图3所示的CMG，它对应ESORE $[((ab)^{\langle 1,2\rangle}+(c)^{\langle 1,2\rangle})(d+c)^{\langle 1,3\rangle}]^{\langle 1,2\rangle}$，当$opt=false$时，省略了标注。

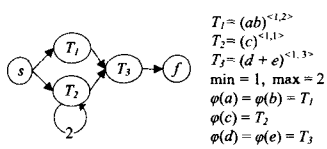

图3 CMG示例

4.2 算法XSDInfer

XSDInfer使用SAX对输入的XML文档(集)进行解析，并在解析的同时进行内容模型的推理。在XML解析过程中，当解析到某个元素e时，XSDInfer首先记录e的上下文相关信息(到根结点的路径p_e)，然后使用XStruct的元素内容模型(element content model)生成算法得到e的一个内容模型实例E_e，最后使用算法eCMGMerge将E_e合并到e的内容模型图CMG_e中。完成XML解析后，CMG已经记录下对应的上下文相关元素的所有内容模型实例的信息，再使用算法eGenerate将CMG转化为内容模型。算法1给出了XSDInfer的具体内容。

算法1 XSDInfer

输入：XML文档(集合)

输出：XSD

1. 使用SAX解析XML的过程中：

i) 当遇到元素e的开始标记时，记录e到根结点的路径p_e，开始记录e的子元素。

ii) 当遇到元素e的结束标记时，生成内容模型实例E_e，调用eCMGMerge(CMG_e, E_e)。用(p_e, CMG_e)更新SM。

2. 当XML解析完毕：

为所有的CMG调用eGenerate(CMG)。

可见，XSDInfer 维护了内容模型的上下文信息。并且，由于在 XML 解析的过程中，XSDInfer 只需要维护 CMG，而不需要存储每个内容模型实例，并且不需要使用文献[3]的 factoring 算法，因此 XSDInfer 在时间消耗与 XStruct 持平的情况下显著减低了内存的消耗。

4.3 合并算法——eCMGMerge

本节给出将内容模型实例合并到 CMG 中的算法 eCMGMerge。如 3.2 节所述，由于内容模型是由若干 Term 连接而成的 ESORE，且对于任意两个 Term T_i 和 T_j，$\sigma(T_i)\cap\sigma(T_j)=\emptyset$，因此对于该 ESORE 对应语言中的任意句子，其中某个字符的每一次出现必然都是由同一个 Term 生成的。根据这个想法，算法 eCMGMerge 的主要思想是将解析过程中得到的每个内容模型实例包含的 Term 信息以及 Term 之间的相对位置信息记录到相应的 CMG 中，同时保证 CMG 中任意两个 Term 不包含相同字符。对于一个要进行合并的内容模型实例 $E=[T_1T_2\cdots T_k]^{\langle \min,\max\rangle}$，称相应 CMG 中与 T_i 包含相同字符的 Term 为 T_i 在 CMG 中的相关 Term，记 T_i 的相关 Term 的集合为 R-Terms(T_i)。按照如下的方法依次处理 E 的每个 Term：若 $T_i(i\in1,2,\cdots,k)$ 中的所有字符均未在 CMG 中出现过，则直接将 T_i 作为一个新结点添加到 CMG 中，并添加相关的边，同时更新 CMG_φ；否则，按照 tTRANS 规则处理 T_i 和 R-Terms(T_i)。

算法 2 eCMGMerge

输入：当前的内容模型图 CMG=$(V,L,s,f,\min,\max,\varphi)$，要合并的内容模型实例 $E=(T_1T_2\cdots T_k)^{\langle \min,\max\rangle}$

输出：合并后的内容模型图 CMG

1. 令 $T_0=s$，$T_{k+1}=f$。
2. //将 E 合并到 CMG 中
   1) $CMG_{min}=MIN\{CMG_{min},E_{min}\}$，$CMG_{max}=MAX\{CMG_{max},E_{max}\}$
   2) for i from 1 to k do
      if $\forall s\in\sigma(T_i),\varphi(s)=$ null then
         i) 将 T_i 加入到 V
         ii) 将(T'_{i-1},T_i)和(T_i,T_{i+1})加入到 L，T'_{i-1}是 T_{i-1}合并后的 Term
         iii) 更新 φ
      else
         i) 根据 φ 找出 R-Terms(T_i)，使用 tTRANS 规则对 T_i 和 R-Terms(T_i)进行处理。
         ii) 更新 φ
   3) 将边(T_k',f)加入 L。

注：MIN 和 MAX 分别为取最小值和最大值的函数。初始时，CMG=$(\{s,f\},\emptyset,MAX_INT,-1,\emptyset)$。其中 MAX_INT 是最大整数。

tTRANS 规则。每条 tTRANS 规则都包含以下两个部分：(a)Term 的处理规则。用于将内容模型实例中待处理的 Term insTerm 与 R-Terms(insTerm)进行合并，或根据 insTerm 对 R-Terms(insTerm)的 Term 进行重构，从而保证 CMG 中任意两个 Term 都不含公共字符，且每个 Term 包含适当数目的字符。(b) CMG 的重构规则。用于在 CMG 中添加、删除 Term 对应的结点，添加、删除或重定向相关的边。

根据 insTerm 和 R-Terms(insTerm)的 Term 的类型，可以大致分为 6 种情况，表 1 给出了每种情况适用的 tTRANS 规则。通过对实际数据的观察，发现大部分属于情况(3)，因此对其进行了更细致的划分。

表 1 各类 Term 适用的 tTRANS 规则

待处理 Term 的类型		适用的 tTRANS 规则
2 个 AndTerm		规则 1
(1) 所有的 Term 都是 OrTerm		规则 2
(2) 既有 AndTerm，也有 OrTerm		规则 3
(3) 所有的 Term 都是 AndTerm	(3.1) R-Terms(insTerm)只含一个 Term cmgTerm，(cmgTerm)$_{max}=1$，且 P_String(insTerm)是 P_String(cmgTerm)的子串	规则 5
	(3.2) R-Terms(insTerm)只含一个 Term cmgTerm，(insTerm)$_{max}=1$，且 P_String(cmgTerm)是 P_String(insTerm)的子串	规则 6
	(3.3) (insTerm)$_{max}\leqslant1$，R-Terms(insTerm)只有一个 Term cmgTerm，且不属于情况(3.1)和(3.2)	规则 7
	(3.4) 其它	规则 4

其中，规则 1 说明如何合并两个 AndTerm，是其它规则的基础。规则 1—规则 4 对 Term 进行合并，规则 5—规则 7 则对 Term 进行拆分重构，以避免一个 Term 包含太多的字符，因为如果 Term 中包含的字符数过多，那么 tTRANS 规则的灵活性和准确度就会下降。下面将详细介绍这些规则并给出示例。在示例中，若 $opt=false$，则省略标注。

规则 1(a)(两个 AndTerm aTerm1，aTerm2 的合并规则) 设合并后的 Term 为 T_{merge}。

1)若 aTerm1 和 aTerm2 中公共字符出现的顺序一致，则

$T_{merge}=(s_1^{opt_1}s_2^{opt_2}\cdots s_m^{opt_m})^{\langle X,Y\rangle}$

式中，$s_1,s_2,\cdots,s_m\in\sigma(aTerm1)\cup\sigma(aTerm2)$。当 $s_i\in\sigma(aTerm1)\cap\sigma(aTerm2)$，且 $opt_{aTerm1}(s_i)=false$，$opt_{aTerm2}(s_i)=false$ 时，$opt_i=false$；其它情况下 $opt_i=true$，$i\in\{1,2,\cdots,m\}$。$X=MIN\{aTerm1_{min},aTerm2_{min}\}$，$Y=MAX\{aTerm1_{max},aTerm2_{max}\}$。$T_{merge}$ 中字符的顺序按照如下方法确定：对于两个字符 s_p 和 s_q($p,q\in\{1,2,\cdots,m\}$)，若 s_p,s_q 都在 $aTerm1$(或 $aTerm2$)中，那么 s_p,s_q 在 T_{merge} 的相对顺序与它们在 $aTerm1$(或 $aTerm2$)的相对顺序相同；否则，假设 $s_p\in T_{aTerm1}$，$s_q\in T_{aTerm2}$，$\exists s_{com}=\sigma(T_{aTerm1})\cap\sigma(T_{aTerm2})$，则 s_p,s_q 在 T_{merge} 中的相对顺序由其在 T_{aTerm1}，T_{aTerm2} 中相对于 s_{com} 的顺序确定。

例如，$T_1=(a\ x\ b\ y^{opt}\ z)^{\langle0,2\rangle}$，$T_2=(d\ x\ y\ z)^{\langle1,1\rangle}$，则

$T_{merge}=(a^{opt}\ d^{opt}\ x\ b^{opt}\ y^{opt}\ z)^{\langle0,2\rangle}$

2)若 aTerm1 和 aTerm2 中公共字符出现的顺序不一致，则

$T_{merge}=(s_1^{opt_1}+s_2^{opt_2}+\cdots+s_m^{opt_m})^{\langle X,Y\rangle}$

式中，$s_1,s_2,\cdots,s_m,opt_i$ 同 1)。$X=MIN\{(size(T_i)-size(T_i[opt=true]))\times(T_i)_{min}\mid T_i=aTerm1,aTerm2\}$，$Y=MAX\{size(T_i)\times(T_i)_{max}\mid T_i=aTerm1,aTerm2\}$，其中 $size(T_i)$ 为 T_i 的字符数，$size(T_i[opt=true])$为 T_i 中 $opt=true$ 的字符数。

例如，$T_1=(y\ a^{opt}\ x)^{\langle1,1\rangle}$，$T_2=(x\ y\ z)^{\langle1,2\rangle}$，则

$T_{merge}=(y+a^{opt}+x+z)^{\langle3,6\rangle}$

规则 2(a)(OrTerm 的合并规则) 设待处理的 Term 为 $T_1,T_2,\cdots,T_k$，都是 OrTerm，合并后的 Term 为 T_{merge}，则

$T_{merge}=(s_1^{opt_1}+s_2^{opt_2}+\cdots+s_m^{opt_m})^{\langle X,Y\rangle}$

式中，$s_1,s_2,\cdots,s_m\in\bigcup_{i=1}^{k}\sigma(T_i)$。当 $s_i\in\bigcap_{j=1}^{k}\sigma(T_j)$，且 $\forall j\in$

$\{1,2,\cdots,k\}$, $opt_{T_j}(s_i)=false$, 则 $opt_i=false$; 其它情况下 $opt_i=true, i\in\{1,2,\cdots,m\}$。$X=\text{MIN}\{(T_l)_{\min}\mid l\in\{1,2,\cdots,k\}\}$, $Y=\text{MAX}\{(T_l)_{\max}\mid l\in\{1,2,\cdots,k\}\}$。

例如, $T_1=(a+b+c)^{\langle 0,1\rangle}$, $T_2=(x+a)^{\langle 1,2\rangle}$, 则

$T_{merge}=(a+b+c+x)^{\langle 0,2\rangle}$

规则 3(a)(AndTerm 与 OrTerm 的合并规则) 设待处理的 Term 为 $T_1,T_2,\cdots,T_k$, 合并后的 Term 为 T_{merge}, 则

$T_{merge}=(s_1^{opt_1}+s_2^{opt_2}+\cdots+s_m^{opt_m})^{\langle X,Y\rangle}$

式中, $s_1,s_2,\cdots,s_m\in\bigcup_{i=1}^{k}\sigma(T_i)$, opt_i 同规则 2(a)。$X=\text{MIN}\{(OTerm_i)_{\min},(size(ATerm_j)-size(ATerm_j[opt=true]))\times(ATerm_j)_{\min}\mid OTerm_i$ 是 $T_1,T_2,\cdots,T_k$ 中的 OrTerm, $ATerm_j$ 是 $T_1,T_2,\cdots,T_k$ 的 AndTerm$\}\}$, $Y=\text{MAX}\{(OTerm_i)_{\max},(ATerm_j)_{\max}\times size(ATerm_j)\mid OTerm_i$ 是 $T_1,T_2,\cdots,T_k$ 中的 OrTerm, $ATerm_j$ 是 $T_1,T_2,\cdots,T_k$ 中的 AndTerm$\}$。

例如, $T_1=(a+b+c)^{\langle 0,1\rangle}$, $T_2=(b\ x)^{\langle 1,2\rangle}$, 则

$T_{merge}=(a+b+c+x)^{\langle 0,4\rangle}$

规则 4(a) 若待处理的 Term 都是 AndTerm, 则使用规则 1 逐个进行合并, 最终得到一个 Term T_{merge}。若合并过程中产生了 OrTerm, 则使用规则 3 将这个 OrTerm 与尚未处理的 AndTerm 进行一次合并。

规则 1—规则 4 的(b)部分相同, 如下:

规则 1(b)—规则 4(b)(添加 T_{merge} 对应的结点) 将 CMG 中与 R-Terms(insTerm)关联的边重定向到 T_{merge}, 若存在反身边, 则记录下反身边的权(重复次数)。删除 CMG 中 R-Terms(insTerm)对应的结点。

例如, CMG 如图 4 所示, R-Terms(insTerm)$=\{T_1,T_3\}$, 则合并后的 CMG 如图 5 所示。

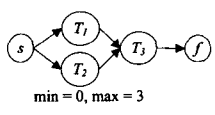

图 4

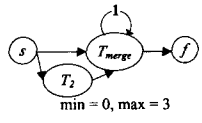

图 5

规则 5(a) 将 cmgTerm 拆分为 Term ST_1,ST_2,ST_3, 使得 $P_String(cmgTerm)=P_String(ST_1)\ P_String(ST_2)\ P_String(ST_3)$, $P_String(ST_2)=P_String(insTerm)$, 且 $(ST_1)_{\min}=(ST_3)_{\min}=0$, $(ST_1)_{\max}=(ST_3)_{\max}=1$, $(ST_2)_{\min}=\text{MIN}\{(cmgTerm)_{\min},(insTerm)_{\min}\}$, $(ST_2)_{\max}=\text{MAX}\{1,(insTerm)_{\max}\}$, $opt_{ST_1}(s)=opt_{ST_3}(s)=opt_{cmgTerm}$, $opt_{ST_2}(s)=opt_{cmgTerm}(s)\mid opt_{insTerm}(s)$。注意, ST_1 和 ST_3 可能不存在。

例如, $cmgTerm=(a\ b^{opt}\ c)^{\langle 1,1\rangle}$, $insTerm=(bc)^{\langle 1,2\rangle}$, 则 $ST_1=(a)^{\langle 0,1\rangle}$, $ST_2=(b^{opt}c)^{\langle 1,2\rangle}$, $ST_3=\text{null}$。

规则 6(a)(规则 6 与规则 5 是对称的) 类似地, 将 insTerm 拆分为 Term ST_1,ST_2,ST_3, 使得 $P_String(insTerm)=P_String(ST_1)P_String(ST_2)P_String(ST_3)$, $P_String(ST_2)=P_String(cmgTerm)$, 且 $(ST_1)_{\min}=(ST_3)_{\min}=0$, $(ST_1)_{\max}=(ST_3)_{\max}=1$, $(ST_2)_{\min}=\text{MIN}\{(cmgTerm)_{\min},(insTerm)_{\min}\}$, $(ST_2)_{\max}=\text{MAX}\{(cmgTerm)_{\max},1\}$, $opt_{ST_1}(s)=opt_{ST_3}(s)=opt_{insTerm}$, $opt_{ST_2}(s)=opt_{cmgTerm}(s)\mid opt_{insTerm}(s)$。$ST_1$ 和 ST_3 可能不存在。

例如, $insTerm=(ab^{opt}cd)^{\langle 1,1\rangle}$, $cmgTerm=(bc)^{\langle 0,2\rangle}$, 则 $ST_1=(a)^{\langle 0,1\rangle}$, $ST_2=(b^{opt}c)^{\langle 0,2\rangle}$, $ST_3=(d)^{\langle 0,1\rangle}$。

规则 5(b)、规则 6(b)(添加 ST_1,ST_2,ST_3 对应的结点) 将 CMG 中指向 $cmgTerm$ 的边重定向到 ST_1, 添加边 $(ST_1,ST_2),(ST_2,ST_3)$, 将以 $cmgTerm$ 为起点的边改为以 ST_3 为起点。删除 $cmgTerm$ 对应的结点。

规则 7(a) 根据 R-Terms(insTerm)对 insTerm 进行拆分。设 $insTerm=(s_1^{opt_1}\cdots s_m^{opt_m})^{\langle x,1\rangle}$, $R\text{-}Terms(insTerm)=\{T_1,T_2,\cdots,T_k\}$, 拆分后的 Term 为 $\{ST_1,ST_2,\cdots,ST_t\}$, 这些 Term 分别属于两种类型 ST_{nType1} 和 ST_{nType2}, 生成方法如下:

对于每一个 $T_r, r\in\{1,2,\cdots,k\}$, 找出 T_r 与 insTerm 的第一个公共字符 s_p 和最后一个公共字符 s_q, 得到一个临时 Term $T_{temp}=(s_p^{opt_p}\cdots s_q^{opt_q})^{\langle x,1\rangle}$, 使用规则 1(a)合并 T_{temp} 和 T_r, 得到新的 Term ST_{nType1}。对于 insTerm 中不能被上述方法处理的字符, 按照它们的相对位置组成新的 Term $ST_{nType2}=(s_1,\cdots,s_n)^{\langle 0,1\rangle}$。

例如, $insTerm=(a\ b\ y\ c\ d\ e\ f)^{\langle 1,1\rangle}$, $R\text{-}Terms(insTerm)=\{(b^{opt}\ x\ c)^{\langle 1,2\rangle},(d)^{\langle 1,2\rangle}\}$, 则结果为 $(a)^{\langle 0,1\rangle}$, $(b^{opt}\ x^{opt}\ y^{opt}\ c)^{\langle 1,2\rangle}$, $(d)^{\langle 1,2\rangle}$, $(ef)^{\langle 0,1\rangle}$。

规则 7(b)(添加 $ST_1,ST_2,\cdots,ST_t$ 对应的结点) 将 CMG 中指向 cmgTerm 的边重定向到 ST_1, 添加边 $(ST_1,ST_2),(ST_2,ST_3),\cdots,(ST_{t-1},ST_t)$, 将以 $cmgTerm$ 为起点的边改为以 ST_t 为起点。删除 $cmgTerm$ 对应的结点。

例如, CMG 如图 4 所示, $R\text{-}Terms(insTerm)=\{T_3\}$, 则合并后的 CMG 如图 6 所示。

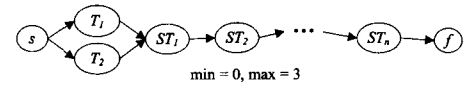

图 6 合并后的 CMG

4.4 内容模型生成算法——eGenerate

完成 XML 解析后, CMG 中已经记录了对应上下文相关元素的所有可能的内容和结构。为了得到内容模型, 需要将 CMG 转化为正则表达式(ESORE)。由 CMG 的构造过程可知, CMG 中结点的拓扑关系反映了 ESORE 中 Term 的位置关系, 下面算法 eGenerate 针对 CMG 的不同拓扑结构分别给出了转化方法。

算法 3 eGenerate

输入: 内容模型图 CMG

输出: 内容模型(ESORE)

1. 删除反身边: 若结点 $v=T_x$ 有权为 n 的反身边, 则用 $v'=(T_x)^{\langle 1,n\rangle}$ 代替 v, 删除反身边。

2. 合并圈: 设圈中的结点为 $v_i=T_i, (i=1,2,\cdots,n)$, 用 $v'=(T_1+T_2+\cdots+T_n)^{\langle 1,\text{unbounded}\rangle}$ 代替 v_i, 删除圈中的边, 重定向关联到 $v_i(i=1,2,\cdots,n)$的边。

3. 合并线性结构: 若存在结点 $v_i=T_i, (i=1,2,\cdots,n)$, 除了 v_1, 每个结点都只有一个前驱 v_{i-1}, 且除了 v_n, 每个结点都只有一个后继 v_{i+1}, 则用 $v'=(T_1T_2\cdots T_n)$替换 v_i, 并重定向相关的边。

4. 合并选择结构: 若存在结点 $v_i=T_i, (i=1,2,\cdots,n)$, 它们具有相同的前驱结点集和后继结点集, 且 $\forall i,j\in\{1,2,\cdots,n\}$, v_i 与 v_j 之间没有边, 则用 $v'=(T_1+T_2+\cdots+T_n)$替换 v_i, 并重定向相关的边。

5. 反复执行步骤 1—步骤 4, 直到 CMG 中除了 s 和 f, 只剩下一个结点 v。CMG 对应的内容模型为 $[v_{\text{esore}}]^{\langle \min,\max\rangle}$

下面以图 3 所示的 CMG 来说明算法 eGenerate 的执行过程。根据步骤 1 删除 T_2 的反身边, 用 $T_2'=(c)^{\langle 1,2\rangle}$ 替换 T_2, 如图 7 所示。根据步骤 4 合并 T_1 和 T_2' 形成的选择结

构，得到的 CMG 如图 8 所示。图 8 中 T_{12} 与 T_3 形成线性结构，根据步骤 3 合并成 T_{123}，如图 9 所示。算法执行完毕，得到内容模型为$[((ab)^{\langle 1,2\rangle}+(c)^{\langle 1,2\rangle})(d+e)^{\langle 1,3\rangle}]^{\langle 1,2\rangle}$。

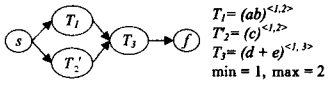

图 7

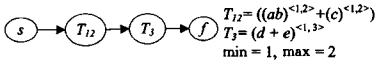

图 8

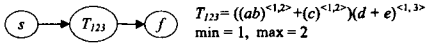

图 9

将所有上下文相关元素的 CMG 都化为内容模型后，可能存在不同的上下文相关元素的内容模型对应相同的 ESORE 的情况。例如图 1 所示的 XML 片段，manager 和 employee 的内容模型都是[name · phone]$^{\langle 1,1\rangle}$，为了使结果 XSD 更简练，可以将[name · phone]$^{\langle 1,1\rangle}$ 作为命名复杂类型 personType，将 manager 和 employee 的类型都指定为 personType，如图 2 所示的 XSD。

5 实验与分析

关于 XSD 提取的研究中，只有 XStruct 和 Trang 是可获得可执行程序的，因此本节中主要在时间/空间消耗及表达能力方面将 XSDInfer 与 XStruct 和 Trang 进行对比分析。由于实际可获得的 XSD 的内容模型在第 1 节所述的表达能力(1)方面基本都是等同于 DTD 的[2]，因此我们手工编写了含有上下文相关内容模型的 XSD 和对应的 XML 文档，用来测试 XSDInfer 对上下文相关内容模型的识别能力。实验结果表明，XSDInfer 能够成功分析出上下文相关的内容模型。

5.1 实验数据

由于实际中带有 XSD 的 XML 文档规模都比较小，考虑到 XStruct 和 Trang 生成的 XSD 的内容模型都不具有上下文相关性，而且我们已经验证过 XSDInfer 对上下文相关内容模型的识别能力，因此选择使用下面两个研究中常用的 XML 数据集：①由 XMark[12] 提供的 XML 生成工具 xmlgen 自动生成的 XML 文档。xmlgen 能够生成 26.5kB 以上任意大小的 XML 文档，这些 XML 都符合同一个模式(auction.dtd)。②XML Data Repository[13]。其不但提供了 XML 数据集，而且提供了相关统计信息，选择使用其中的 SwissProt 和 Treebank。

5.2 可扩展性

实验环境为 Pentium D CPU 3.40GHz，1.50G 内存，Windows XP 操作系统，JDK 1.6。测试对象是进程占用的内存量，因此结果数据中包含了 JVM 和进程本身的空间开销。下面分别给出对于 xmlgen 自动生成 XML 数据集及 SwissProt，Treebank 数据集的测试结果。

5.2.1 xmlgen 自动生成的数据集

将 xmlgen 的参数分别设置为 0.02，0.035，0.04，0.1，0.5，1，5，10，生成了大小为 2.3MB，4MB，4.61MB，11.3MB，56.2MB，113MB，568MB，1137MB 的 XML 文档。由于 Trang 无法处理 xmlgen 生成的 XML 文档(抛出异常)，因此下面只给出 XSDInfer 和 XStruct 在时间消耗和内存占用方面的对比结果，如表 2 和图 10 所示。

表 2 XSDInfer 与 XStruct 的时间对比

s \ MB	2.3	4	4.61	11.3	56.2	113	568	1137
XSDInfer	0.46	0.67	0.78	1.56	6.75	12.7	64.5	137
XStruct	0.45	0.67	0.79	1.65	7.32	13.0	59.7	124

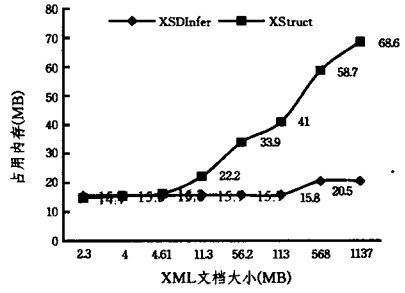

图 10 XSDInfer 与 XStruct 占用内存对比

可以看出，对于大规模的 XML 文档，XSDInfer 与 XStruct 相比，在耗时基本相同的情况下大大降低了内存的占用，实验结果验证了上文的分析。图 10 也表明，对于 xmlgen 数据集，XSDInfer 的内存占用量是比较稳定的，这是因为 XSDInfer 只需要为每个上下文相关元素维护一个 CMG。当 xmlgen 生成的 XML 文档规模变大时，分析得到的 CMG 的规模并没有显著变化。而对于 XStruct，由于要存储所有不同的内容模型实例，因此占用的内存会显著增加。

5.2.2 SwissProt 和 Treebank 数据集

表 3 给出了 XML Data Repository 提供的 SwissProt 和 Treebank 的统计信息。

表 3 SwissProt 和 Treebank 的统计信息

	大小(MB)	元素数	最大深度	平均深度
SwissProt	109	2977031	5	3.55671
Treebank	82	2437666	36	7.87279

在默认的 JVM 缓存大小下，XStruct 会因堆栈溢出而无法处理这两个数据集。这是因为 XStruct 使用了递归算法 factoring，导致 XStruct 无法处理较大规模的、深度嵌套的 XML 文档。Trang 能够处理 SwissProt，在时间和空间消耗方面表现得也比较好，但是 Trang 无法处理 Treebank(抛出异常)。XSDInfer 的实验结果如表 4 所列。可以看出，对于较大规模、深度嵌套的 XML 文档，XSDInfer 依然具备良好的处理能力。

表 4 XSDInfer 在 SwissProt 和 Treebank 的结果

	SwissProt	Treebank
时间(s)	23.7	38.9
占用内存(MB)	15.7	16.0

5.3 表达能力

由于 Trang 基本无法处理实验中的 XML 数据集，因此本节只给出 XSDInfer 和 XStruct 根据 xmlgen 自动生成的 XML 文档(0.025)提取得到的部分内容模型的对比情况，如表 5 所列。其中第 1 列给出元素的名字和在 XML 文档中出现的次数，第 2 列分别给出了相应的结果。当 min 和 max 都为 1 时，省略了标注。为了便于比较，使用“?”代替 XSDInfer 结果中的“opt”标记。值得说明的是，第 2 列中也给出了 xml-

gen 所提供的 auction. dtd 中的内容模型。在这里我们不考虑上下文相关性，因此可以参照 auction. dtd 的内容模型来检验 XSDInfer 结果的正确性。

表 5　表达能力对比

元素(次数) \ 内容模型	1. auction. dtd 2. XSDInfer 3. XStruct
item (543)	1. a b c d e f g+ h 2. (a b c d e f)(g)⟨1,unbounded⟩(h) 3. (a b c d e f(g)⟨1,unbounded⟩h)+(a b c d e f g h)
person (637)	1. (a b c? d? e? f? g? h?) 2. (a b)(c? d? e? f? g? h?)⟨0,1⟩ 3. 47 个字符的表达式
address (304)	1. a b c d? e 2. a b c d? e 3. a b c((d e)+e)
open_auction (300)	1. a b? c* d e? f g h i j k 2. (a)(b)⟨0,1⟩(c)⟨0,unbouned⟩(d e? f g h i j k) 3. 94 个字符的表达式
profile (312)	1. a* b? c? d e? 2. a⟨0,unbounded⟩ b⟨0,1⟩ c⟨0,1⟩(d e?) 3. 31 个字符的表达式

从表 5 可以看出，XSDInfer 能够得到比较准确的结果，并在简洁性和可读性方面都优于 XStruct。

结束语　本文提出了一种基于内容模型图的 XSD 提取方法 XSDInfer，实验结果表明，与同类方法相比，XSDInfer 具有更好的可扩展性和表达能力。

在未来的工作中，我们将考虑对命名空间的支持。另外，虽然 tTRANS 规则给出了实际中常见情况下的处理方法，但是在时间和空间代价允许的前提下，我们仍然可以考虑对各种情况进行更加细致的划分，从而得到更精确的结果。

参 考 文 献

[1] Sun. JAXB[EB/OL]. http://java. sun. com/webservices/jaxb

[2] Bex G J, Neven F, Vansummeren S. Inferring XML schema definitions from XML data[C]// Proceedings of the 33rd International Conference on Very Large Data Bases. Vienna, Austria: VLDB Endowment, 2007: 998-1009

[3] Hegewald J, Naumann F, Weis M. XStruct: efficient schema extraction from multiple and large XML documents[C]// Proceedings of 22nd International Conference on Data Engineering Workshops. Atlanta, GA, USA, 2006: 81

[4] Altova XMLSpy[EB/OL]. http://www. altova. com/products/xmlspy/xmlspy. html

[5] Stylus Studio[EB/OL]. http://www. stylusstudio. com/

[6] Trang C J. Multi-format schema converter based on RELAX NG [EB/OL]. Thai Open Source Software Center Ltd, 2003. http://www. thaiopensource. com/relaxng/trang. html

[7] Garofalakis M, Gionis A, Rastogi R, et al. XTRACT: learning document type descriptors from XML document collections[J]. Data Mining and Knowledge Discovery, 2003, 7(1): 23-56

[8] Min J, Ahn J, Chung C. Efficient extraction of schemas for XML documents[J]. Information Processing Letters, 2003, 85(1): 7-12

[9] Bex G, et al. Inference of concise DTDs from XML data[C]// Proceedings of the 32nd International Conference on Very Large Data Bases. Seoul, Korea: VLDB Endowment, 2006: 115-126

[10] Martens W, Neven F, Schwentick T. Simple off the shelf abstractions for XML Schema[J]. ACM SIGMOD Record, 2007, 36(3): 15-22

[11] Thompson H, et al. XML schema part 1: Structures[S]. W3C Recommendation, 2001

[12] XMark[EB/OL]. http://monetdb. cwi. nl/xml/

[13] Miklau G, Suciu D. XML data repository[EB/OL]. http://www. cs. washington. edu/research/xmldatasets/

(上接第 158 页)

[3] Feng X, Ni Z, Shao Z, et al. An open framework for foundational proof-carrying code[C]// Proceeding of ACM SIGPLAN International Workshop on Types in Language Design and Implementation. 2007: 67-78

[4] King J C. Symbolic Execution and Program Testing[J]. Communications of the ACM, 1976, 19(7): 385-394

[5] Zhang Jian. Symbolic execution of program paths involving pointer and structure variables[C]// 4th International Conference on Quality Software(QSIC'04). 2004: 87-92

[6] Xu Z, Zhang J. A test data generation tool for unit testing of C programs[C]// International Conference on Quality Software (QSIC'06). 2006: 63-74

[7] Cadar C, Dunbar D, Engler D. KLEE: Unassisted and Automatic Generation of High-coverage Tests for Complex Systems Programs[C]// USENIX Symposium on Operating Systems Design and Implementation (OSDI 2008). San Diego, December 2008: 33-45

[8] Cadar C, Dunbar D. EXE: Automatically generating inputs of death[J]. ACM Transactions on Information and System Security, 2008, 12(2): 26-43

[9] Necula G C, McPeak S, Rahul S, et al. CIL: Intermediate language and tools for analysis and transformation of C programs [C]// International Conference on Compiler Construction. March 2002: 67-83

[10] Ganesh V, Dill D L. A decision procedure for bit-vectors and arrays[C]// Proceeding of the 19th International Conference on Computer Aided Verification. 2007: 87-96

[11] Wanger D, Foster J S, Brewer E A, et al. A first step towards automated detection of buffer overrun vulnerabilities[C]// 7th Network and Distributed System Security Symposium. California, USA, 2000: 23-45

[12] Godefroid P, Klarlund N, Sen K. DART: directed automated random testing[C]// Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation. New York, USA, 2005: 213-223

[13] Sen K, Marinov D, Agha G. CUTE: a concolic unit testing engine for C[C]// Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, USA, 2005: 263-272