Hans Journal of Data Mining 数据挖掘, 2022, 12(4), 334-350
Published Online October 2022 in Hans. http://www.hanspub.org/journal/hjdm
https://doi.org/10.12677/hjdm.2022.124032

Hans汉斯

基于示例编程的层次模型到关系模型的数据转换

周晓楠，李　贵，李征宇

沈阳建筑大学，辽宁 沈阳

收稿日期：2022年9月2日；录用日期：2022年10月2日；发布日期：2022年10月12日

摘　要

将多个数据源中的数据结合起来并统一存储，建立数据仓库的过程是web数据集成中的一个重要步骤。数据集成通过数据转换从而达到集成，主要解决数据的分布性和异构性的问题。许多应用程序使用层次结构存储和传输数据，这种基于树结构的层次模型非常适合底层数据，因此分层数据格式很流行用于导出数据并在不同应用程序之间传输数据。为了便于存储和查询通常需要将此类层次结构数据转换为关系表示，但由于层次结构数据和关系结构数据的特点以及需要处理的数据源可能很大，给这一转换过程带来了不少的工作量。为了解决这个问题，本文采用了一种基于示例编程的方法，用于将层次结构的文档迁移到关系格式。通过提出一种程序合成算法将合成关系表的任务分解为列提取和行提取这两个子任务，从输入输出示例学习目标转换，实现XML文档或JSON文档转换为关系表。实验结果表明，本文的方法可以为从层次结构数据到关系数据的转换任务生成所需的程序，实现数据集中的数据转换。

关键词

Web数据集成，数据转换，层次模型，示例编程

Data Transformation from Hierarchical Model to Relational Model Based on Example Programming

Xiaonan Zhou, Gui Li, Zhengyu Li

Shenyang Jianzhu University, Shenyang Liaoning

Received: Sep. 2nd, 2022; accepted: Oct. 2nd, 2022; published: Oct. 12th, 2022

Abstract

The process of combining and storing data from multiple data sources and establishing a data

文章引用: 周晓楠, 李贵, 李征宇. 基于示例编程的层次模型到关系模型的数据转换[J]. 数据挖掘, 2022, 12(4): 334-350. DOI: 10.12677/hjdm.2022.124032

warehouse is an important step in web data integration. Data integration achieves integration through data transformation, and mainly solves the problems of data distribution and heterogeneity. Many applications store and transfer data using a hierarchical structure. This tree-based hierarchical model is well suited to the underlying data, so hierarchical data formats are popular for exporting data and transferring data between applications. In order to facilitate storage and query, it is usually necessary to convert such hierarchical data into relational representation. However, due to the characteristics of hierarchical data and relational data and the large data sources that need to be processed, this conversion process brings a lot of difficulties. workload. To address this issue, this paper adopts an example-based programming approach for migrating hierarchically structured documents to a relational format. By proposing a program synthesis algorithm, the task of synthesizing relational tables is decomposed into two sub-tasks of column extraction and row extraction, learning target conversion from input and output examples, and converting XML documents or JSON documents into relational tables. The experimental results show that the method in this paper can generate the required programs for the transformation task from hierarchical data to relational data, and realize the data transformation in the dataset.

Keywords

Web Data Integration, Data Conversion, Hierarchical Model, Example Programming

1. 引言

许多应用程序使用层次结构格式(例如 XML 或 JSON 文档)存储和传输数据。这种层次结构的数据模型非常适合本质上是分层的底层数据。此外，由于 XML 和 JSON 文档不仅包含数据还包含描述数据属性信息的元数据，因此，这类文档具有自我描述性和可移植性，很大程度提高了以不同的操作系统去使用相同数据集或文件的可能性。由于这些原因，分层的数据格式很流行用于导出数据并在不同应用程序之间传输它们。

尽管分层数据模型很方便，但仍有许多情况需要将它们转换为关系格式。例如，存储在 XML 文档中的数据可能需要由与关系数据库交互的现有应用程序进行查询。此外，因为分层数据模型通常不太适合高效的数据查询，当查询性能很重要时，将分层数据格式转换为关系格式是很有必要的。但在转换过程中，存在以下问题：

1) 由于源数据和目标数据的表示完全不同，所需的转换通常比结构相似的数据之间的转换更复杂。

2) 由于目标表中的每一行都对应着输入层次结构树中节点之间的关系，所以合成关系表过程中需要发现树节点之间的这些“隐藏链接”。

因此，本文采用了一种基于示例编程的方法，由一组简单的输入输出示例说明所需要的转换，这种方法通过学习生成所需任务的转换程序，并可在尽可能少的用户操作下实现将 XML 文档或 JSON 文档转换为关系数据表。

本文提出一种程序合成算法来解决层次结构数据向关系结构数据转换过程中存在的问题，该算法将层次结构数据转换成关系表的任务分解为两个子问题，列学习和行学习。目的是分别学习列和行的构造逻辑：

1) 列学习提取逻辑：给定关系表中的一个属性，本文的方法首先合成一个程序来提取对应于该属性的树节点。换句话说，本文首先忽略不同树节点之间的关系并单独构造每一列。然后取所构造列的叉积会产生一个过度近似目标表的表，这个表中包含多于目标表中元组个数的额外元组。

2) 行学习提取逻辑：由于在上一阶段学习的程序产生了一个与目标关系表过度近似的表，本文算法的这一阶段合成了一个过滤程序，该程序可以过滤掉在上一阶段产生的没有实际意义的“虚假”元组。本质上，算法的第二阶段发现了原始层次结构中不同节点之间的“隐藏链接”。

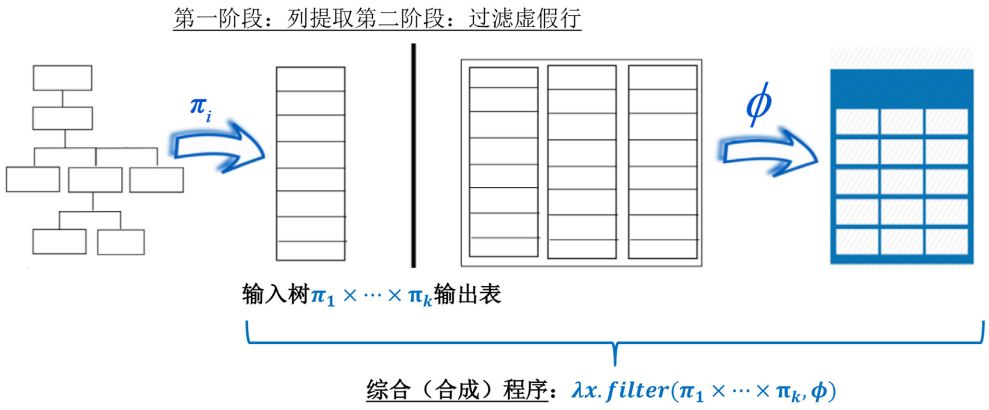


Figure 1. Schematic diagram of the synthesis algorithm
图 1. 合成算法的示意图

图 1 中显示了本文合成算法的示意图。给定输入树 T 和具有 k 列的输出表 R，本文的方法首先学习 k 个不同的列提取程序 $\pi_1,\cdots,\pi_k$，其中每个列提取程序 π_i 从 T 中提取存储在 R 的第 i 列中的数据。本文的合成算法然后通过将每个 π_i 应用于输入树并取其叉积来构造一个中间表。因此，在第一阶段生成的中间表 $\pi_1(T)\times\cdots\times\pi_k(T)$ 是可能包含比输出表 R 更多元组的过度近似目标表。在下一阶段，本文的方法学习一个谓词 φ，可用于从中间表中准确过滤掉虚假元组。因此，算法合成的程序形式表达为 $\lambda x.filter(\pi_1\times\cdots\times\pi_k,\phi)$。此外，由于合成的程序不应该过度拟合用户提供的示例，本文的方法通过学习生成与用户提供的输入输出示例一致的最简单的程序。

本文引用了实际项目中的城市房地产大数据，该数据是利用网络爬虫从某市房地产交易备案信息网站所爬取，并经过数据集成处理的真实数据。该实例展示了某市部分房地产楼盘的信息(其中包含有关部分楼盘的信息以及它们之间的价格关系)。接下来将使用该实例对本文的方法进行一个总体的验证。假设一个用户想要将图 2(a)这个 XML 文档转换成图 2(b)所示的关系表。这种转换非常重要，因为 XML 文件将这些信息存储为从每个楼盘到与之比较房价的楼盘列表的映射，其中每个对比楼盘都由他们的 cid 表示。相反，所需的表将该信息存储为元组(A, B, n)，表示名为 A 的楼盘与名为 B 的楼盘售价比较结果是 pid 为 n 的楼盘售价更低。

假设原始 XML 文件比图 2(a)中所示的文件大得多，因此用户决定通过提供图 2 中的输入输出示例来自动完成 XML 到关系表的转换任务，并自动合成所需的数据转换程序。

工作主要包括：

1) 引用了一种基于示例编程的方法，用于将层次结构文档迁移到关系格式。

2) 引用了一种从树到表转换的领域特定语言，它便于表达层次结构和关系结构之间的转换，可以实现许多现实世界的数据转换任务。

```
〈城市〉〈楼盘 pid=“1” P_name=“米兰颂”〉〈楼盘售价对比〉
〈对比楼盘 cid=“2” 较低价pid=” 1”〉〈/对比楼盘〉
〈对比楼盘 cid=“3” 较低价pid=” 3”〉〈/对比楼盘〉
〈/楼盘售价对比〉〈/楼盘〉
〈城市〉〈楼盘 pid=“2” P_name=“招商公园1872”〉〈楼盘售价对比〉
〈对比楼盘 cid=“3” 较低价pid=” 3”〉〈/对比楼盘〉
〈对比楼盘 cid=“1” 较低价pid=” 1”〉〈/对比楼盘〉
〈/楼盘售价对比〉〈/楼盘〉
〈城市〉〈楼盘 pid=“3” P_name=“汇置尚都”〉〈楼盘售价对比〉
〈对比楼盘 cid=“4” 较低价pid=” 4”〉〈/对比楼盘〉
〈/楼盘售价对比〉〈/楼盘〉
〈城市〉〈楼盘 pid=“4” P_name=“华强芯北上”〉〈楼盘售价对比〉
〈对比楼盘 cid=“1” 较低价pid=” 4”〉〈/对比楼盘〉
〈/楼盘售价对比〉〈/楼盘〉〈/城市〉
```

(a) 输入 XML

(pid)楼盘	(cid)售价对比	较低价楼盘
(1)米兰颂	(2)招商公园 1872	1
(1)米兰颂	(3)汇置尚都	3
(2)招商公园 1872	(1)米兰颂	1
(2)招商公园 1872	(3)汇置尚都	3
(3)汇置尚都	(4)华强芯北上	4
(4)华强芯北上	(1)米兰颂	4

(b)输出关系表

Figure 2. Example of input and output
图 2. 输入输出实例

3) 提出一种程序合成算法，将关系表合成任务分解，分别学习关系表行和列的构造逻辑。首先通过将每个列提取器应用于输入树并取其叉积生成中间表，然后通过学习合适的谓词过滤掉中间表中的虚假元组。最后生成一个可执行的扩展样式表语言 XSLT (eXtensible Stylesheet Language)程序。

4) 通过完成城市房地产大数据实际项目中数据转换，进行实验，证明本文方法的实用性和有效性。

2. 相关研究

将分层结构化文档转换为关系格式的问题是数据转换的一种形式，其目标是将源模式的数据转换为目标模式的数据实例[1]。由于手动执行此类转换的难度，在自动化数据转换任务方面已经开展了大量工作[2] [3] [4] [5] [6]。Clio [2]推广的一种常用方法将数据转换任务分解为两个独立的阶段：模式映射和转换程序生成。模式映射描述了源模式和目标模式之间的关系，通常以声明性逻辑约束的形式指定，例如GLAV (Global-Local-As-View)约束[7]。给定一个模式映射，第二个程序生成阶段将该映射“转换”为可执行代码[3]。因为模式映射的手动构建需要数据架构师付出不小的努力，所以在从用户提供的各种非正式规范中自动推断此类映射方面已经开展了大量工作。

例如，用户可以通过在包含相关数据的元素之间画线来指定元素对应关系[3] [8] [9] [10] [11] [12]。最近，数据转换示例作为用户和系统之间的一种通信方式变得越来越流行。这种基于示例的方法可以根据示例的用途大致分为两类。一种方法使用示例来指定所需的模式映射[3] [5] [6] [13]。以上这些方法都使用 GLAV 作为模式映射语言，因此只能处理源模式和目标模式都是关系的情况。另一种方法使用示例来帮助用户理解和改进生成的模式映射[14] [15]。

本文可以看作是第一种方法的实例化，但在三个方面不同于以前的方法。首先，可以合成将层次结构数据转换为关系数据的程序。其次，设计了一种领域特定语言作为中间语言，它可以表达层次和关系格式之间的映射。最后，合成算法结合了有限自动机和谓词学习，以实现高效的映射生成。

有大量关于使用关系数据库管理系统存储和查询 XML 和 JSON 文档的研究[16]-[24]。这些研究首先将文档的层次结构转换为平面的关系模式，接下来，将 XML 文档“分解”并加载到关系表中。最后，将 XML 查询转换为相应的 SQL 查询。这些研究方法的目标与本文不同，它们的目的是利用关系数据库管理系统有效地完成有关 XML 文档的查询。而本文的目的是完成关于底层数据所需关系表示的 SQL 查询。此外，本文实现层次结构到关系结构的转换方法也不同于现有的 XML 分解技术，现有的技术可以大致分为两类，即以结构为中心的方法和以模式为中心的方法。第一种方法依赖于 XML 文档结构来指导映射过程[21] [25]，第二种方法利用模式信息和注释来导出映射[23] [26] [27]。

而本文在映射过程中使用输入层次结构和输出关系结构实例的方法。此外，设计的领域特定语言可以表达丰富的映射程序类别，并且在本文的合成算法中，基于有限自动机和谓词学习的技术的结合不同于以前的层次结构到关系结构的数据转换方法。

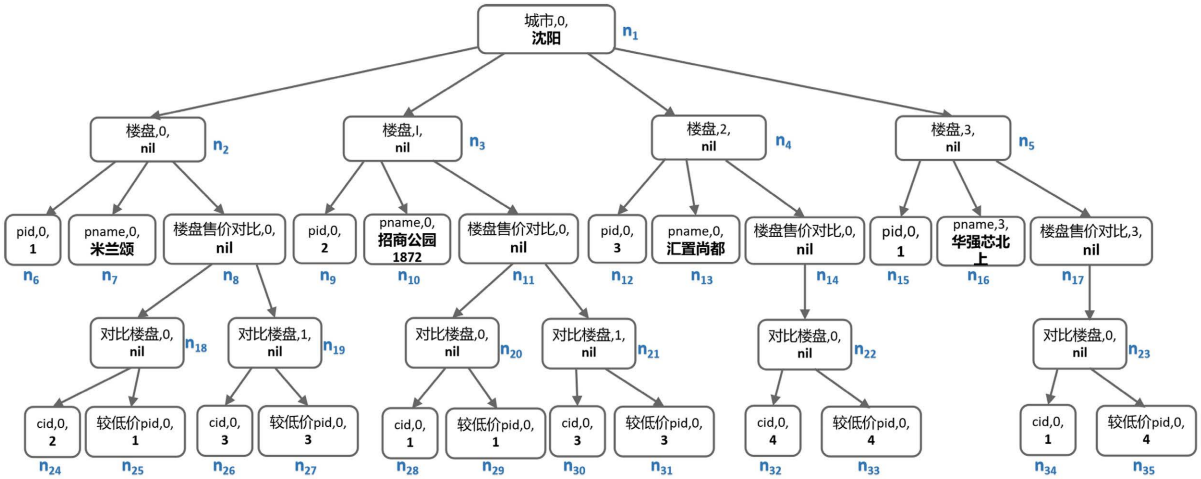
3. 基于实例的数据转换类型

3.1. 层次树模型

为了实现统一表示，本文使用层次树对层次结构文档(例如 XML、HTML 和 JSON)进行建模，这是文献[28]中使用的数据结构的一种变体。

定义 1：层次树(Hierarchical Data Tree)中 τ 是表示为二元组 $\langle V, E \rangle$ 的有根树，其中 V 是一组节点，而 E 是一组有向边。节点 $v \in V$ 是一个三元组(tag, pos, $data$)，其中 tag 是与节点 v 关联的标签，pos 表示 v 是 V 的父节点下带有标签 tag 的第 pos 个元素，$data$ 是对应存储在节点 v 的数据。

给定层次树中的节点 $n = (t, i, d)$，符号 $n.tag$、$n.pos$ 和 $n.data$ 分别表示 t、i 和 d。符号 $isLeaf(n)$来表示 n 是层次树的叶节点。假设只有叶节点包含数据；因此，对于任何内部节点(t, i, d)，本文有 $d = nil$。最后，给定一个层次树 τ，用 $\tau.root$ 来表示 τ 的根节点。



(a) 输入 XML 的层次树表示

$\pi_{11}(\{n_1\})$	$\pi_{21}(\{n_1\})$	$\pi_{31}(\{n_1\})$
n_7	n_7	n_{25}
n_7	n_7	n_{27}
n_7	n_7	n_{29}
n_7	n_7	n_{31}
n_7	n_7	n_{33}
n_7	n_7	n_{35}
n_7	n_{10}	n_{25}
n_7	n_{10}	n_{27}
…		
n_{16}	n_{16}	n_{31}
n_{16}	n_{16}	n_{33}
n_{16}	n_{16}	n_{35}

(b) 中间表

Figure 3. Column extraction in the instance
图 3. 实例中的列提取

```
{ “城市” : { “楼盘” : [
    { “pid” :1, “P_name” : 米兰颂, “楼盘售价对比” :
            { “对比楼盘” : [{ “cid” :2, “较低价pid” :1}, { “cid” :3, “较低价pid” :3}]}},
    { “pid” :2, “P_name” : 招商公园1872, “楼盘售价对比” :
            { “对比楼盘” : [{ “cid” :3, “较低价pid” :3}, { “cid” :1, “较低价pid” :1}]}},
    { “pid” :3, “P_name” : 汇置尚都, “楼盘售价对比” :
            { “对比楼盘” : [{ “cid” :4, “较低价pid” :4}]}},
    { “pid” :4, “P_name” : 华强芯北上, “楼盘售价对比” :
            { “对比楼盘” : [{ “cid” :1, “较低价pid” :4}]}]}]}
```

Figure 4. Example in JSON file
图 4. JSON 文件中的例子

XML 文件作为层次结构数据，可以将 XML 文件表示为层次树，其中节点对应于 XML 中的元素。从节点 v'到节点 $v=(t,i,d)$ 的边表示 v 的 XML 元素 e 直接嵌套在由 v'表示的元素 e'内。此外，由于节点 v 包含标签 t 和位置 i，这意味着元素 e 是标签为 t 的元素 e'的第 i 个孩子。本文还将 XML 属性和文本内容建模为嵌套元素，这种设计选择允许层次树表示包含嵌套元素、属性和文本内容组合的元素。

示例 1：图 3(a)显示了图 2(a)中 XML 文件的层次树表示。展现了属性如何在层次树中表示为嵌套元素。

JSON 文档作为层次结构数据，将数据存储为一组嵌套的键值对。可以通过以下方式将 JSON 文件建模为层次树：层次树中的每个节点 $n=(t,i,d)$ 对应于 JSON 文件中的键值对 e，其中 t 表示键，d 表示值。由于 JSON 文件中的值可以是数组，因此位置 i 对应于数组中的第 i 个条目。例如，如果 JSON 文件将键 k 映射到数组[13] [18] [28]，则层次树中包含的形式为$(k, 0, 18)$，$(k, 1, 45)$，$(k, 2, 32)$。从 n'到 n 的边，n 表示的键值对 e 嵌套在 n'表示的键值对 e'内。

示例 2：图 4 显示了与图 3(a)中的层次树表示相对应的 JSON 文档。其中 JSON 对象和数组表示为数据为 nil 的内部节点。对于给定的节点 n，除非 n 的父节点对应于一个数组，否则 $n.pos = 0$。

3.2. 领域特定语言

在本节中，将介绍用于实现从层次树到关系表的转换的领域特定语言(Domain-Specific Language)。本文将关系表表示为一组元组，其中每个元组都使用列表表示。给定一个关系表 R，使用符号 $column(R, i)$ 来表示 R 的第 i 列，并且可以互换使用术语“关系”和“表”。

$$
\begin{gathered}
Program\ P \coloneqq \lambda\tau.\, filter(\psi, \lambda t, \phi) \\
Table\ Extractor\ \psi \coloneqq (\lambda s.\pi)\ \{root(\tau)\} | \psi_1 \times \psi_2 \\
\text{Column Extractor}\ \pi \coloneqq s \mid children(\pi, tag) \\
\mid pchildren(\pi, tag, pos) \\
\mid descendants(\pi, tag) \\
\text{redicate}\ \phi \coloneqq ((\lambda n, \varphi)\ t[i]) \trianglelefteq c \\
\mid ((\lambda n, \varphi_1)\ t[i]) \trianglelefteq ((\lambda_n, \varphi_2)\ t[j] \\
\mid \phi_1 \wedge \phi_2 | \phi_1 \vee \phi_2 | \neg\phi \\
\text{Node Extractor}\ \varphi \coloneqq n \mid parent(\varphi) | child(\varphi, tag, pos)
\end{gathered}
$$

Figure 5. Domain-specific language grammar
图 5. 领域特定语言语法

图 5 为领域特定语言的语法。这里 τ 表示输入树，t 是绑定到 τ 中的节点元组，n 表示 τ 中的节点。此外，i 和 j 是整数，c 是一个常数值(字符串、整数等)。$t[i]$给出 t 中的第 i 个元素。

图 6 给出了领域特定语言的语义。在解释领域特定语言中的结构之前，重要的一点是领域特定语言的目的是实现表达能力和综合效率之间的良好平衡。也就是说，虽然 DSL 可以表达在实践中出现的一大类树到表的转换，但它的目的是使层次结构到表结构自动化合成变得可行。

$$[\![filter(\psi,\lambda t,\phi)]\!]_T = \{(n_1.data,\cdots,n_k.data)|t \in [\![\psi]\!]_T, t=(n_1,\cdots,n_k), [\![\phi]\!]_{t,T} = True\ \}$$

$$[\![\psi_1 \times \psi_2]\!]_T = \{(n_1,\cdots,n_k,n_1{}',\cdots,n_l{}')|(n_1,\cdots,n_k) \in [\![\psi_1]\!]_T, (n_1{}',\cdots,n_k{}') \in [\![\psi_2]\!]_T\ \}$$

$$[\![(\lambda s.\pi)\{root(\tau)\}]\!]_T = \{n\,|n \in [\![\pi]\!]_{s,T}, s=\{T.root\}\ \}$$

$$[\![x]\!]_{s,T} = s\ \ where\ s\ is\ a\ set\ of\ nodes\ in\ T$$

$$[\![children(\pi,tag)]\!]_{s,T} = \{n'|n \in [\![\pi]\!]_{s,T}, (n,n') = E\ , n'.tag = tag\}\ where\ T=(V,E)$$

$$[\![pchildren(\pi,tag,pos)]\!]_{s,T} = \{n'|n \in [\![\pi]\!]_{s,T}, (n,n') = E\ , n'.tag = tag, n'.pos\ = pos\}\ where\ T=(V,E)$$

$$[\![descendants(\pi,tag)]\!]_{s,T} = \{n_z|n_1 \in [\![\pi]\!]_{s,T}, \exists\{n_2,\cdots,n_{z-1}\}\subset V\ \mathrm{s.t.}\ \forall 1 \le x < z.(n_x,n_{x+1}) \in E\ , n_z.tag = tag\}$$
$$where\ T=(V,E)$$

$$[\![((\lambda n.\varphi)\ t[i]) \trianglelefteq c]\!]_{t,T} = n'.data \trianglelefteq c\ \ where\ t=(n_1,\cdots,n_l)\ and\ n' = [\![\varphi]\!]_{n_i,T}$$

$$[\![((\lambda n.\varphi_1)\ t[i]) \trianglelefteq ((\lambda n.\varphi_2)\ t[j])]\!]_{t,T} = \begin{cases} n'_1.data \trianglelefteq n'_2.data & if\ n'_1\ and\ n'_2\ are\ both\ leaf\ nodes\ of\ T \\ n'_1 = n'_2\ \ if \trianglelefteq is\ "=", & and\ neither\ n'_1\ nor\ n'_2\ is\ a\ leaf\ node\ of\ T \\ & False\ \ otherwise \end{cases}$$

$$[\![\phi_1 \wedge \phi_2]\!]_{t,T} = [\![\phi_1]\!]_{t,T} \wedge [\![\phi_2]\!]_{t,T}$$

$$[\![\phi_1 \vee \phi_2]\!]_{t,T} = [\![\phi_1]\!]_{t,T} \vee [\![\phi_2]\!]_{t,T}$$

$$[\![\neg\phi]\!]_{t,T} = \neg[\![\phi]\!]_{t,T}$$

$$[\![n]\!]_{n,T} = n$$

$$[\![parent(\varphi)]\!]_{n,T} = \begin{cases} n'\ if(n',[\![\varphi]\!]_{n,T}) \in E \\ \perp\ otherwise \end{cases}\ where\ T=(V,E)$$

$$[\![child(\varphi,tag,pos)]\!]_{n,T} = \begin{cases} n'\ if([\![\varphi]\!]_{n,T},n') \in E\ and\ n'.tag = tag\ and\ n'.pos = pos \\ \perp\ otherwise \end{cases}$$
$$where\ T=(V,E)$$

Figure 6. Semantics of Domain-Specific Languages
图 6. 领域特定语言的语义

领域特定语言的结构遵循本文将问题分解为两个单独的列和行提取操作的综合方法。程序 P 的形式为 $\lambda\tau.filter(\psi,\lambda t.\phi)$，其中 τ 是输入层次树，ψ 是从 τ 中提取关系 R'的表提取器。如第 1 节所述，提取的表 R'过度近似目标表 R。因此，过滤器构造使用谓词 ϕ 过滤掉 R'中未出现在 R 中的元组。

表提取器 ψ 通过取多列的叉积来构建表，其中每列中的一个条目是指向输入层次树中节点的“指针”。每列是通过将列提取器 π 应用于输入树的根节点来获得的。列提取器 π 将一组节点和一个层次树作为输入，并返回一组层次树节点。列提取器是递归定义的，可以相互嵌套：基本情况简单时返回节点的输入集，递归情况使用提取函数 *children*、*pchildren* 和 *descendants* 从每个输入节点中提取其他节点。其中，*children* 提取具有给定标签的所有子项，*pchildren* 生成具有给定标签和指定位置的所有子项，*descendants* 返回具有给定标签的所有后代。每个构造的语义在图 6 中给出。

现在介绍可以在过滤器构造中使用的谓词 ϕ。没有布尔连接词的原子谓词是通过将存储在层次树节点中的数据与常量 c 进行比较或与存储在另一个树节点中的数据进行比较来获得的。特别是，谓词使用节点提取器 φ 以树节点作为输入并返回另一个树节点。与列提取器类似，节点提取器是递归定义的，并且可以相互嵌套。节点提取器允许访问父节点和子节点，因此可用于从给定的输入节点中提取层次树内的任意节点(图 6 给出了语义)。

回到谓词的定义，ϕ 接受一个元组 t 并计算为一个布尔值，指示 t 是否应该保留在输出表中。最简单的谓词 $((\lambda n.\varphi)t[i]) \trianglelefteq c$ 首先提取 t 的第 i 个条目 n，然后使用节点提取器 φ 从 n 中获取另一个树节点 n'如果 $n'.data \trianglelefteq c$ 为真，则此谓词的计算结果为真。$((\lambda n.\varphi_1)t[i]) \trianglelefteq ((\lambda n.\varphi_2)t[j])$ 的语义是相似的，它比较了存储在两个树节点 n，n'的值。如果 n，n'都是叶节点，那么检查关系 $n.data \trianglelefteq n'.data$ 是否满足。如果它

们是内部节点并且运算符$\trianglelefteq$是相等的，那么检查n，n'是否指的是同一个节点。否则，谓词评估为假。使用标准布尔连接词$\wedge$，$\vee$，$\neg$可以获得更复杂的谓词。

```
<Dependency><Object id="8">A
  <Object id="13"> B <Object id="18"> D
    <Object id="23"> F </Object>
    <Object id="28"> G </Object> </Object> </Object>
  <Object id="20"> C <Object id="24"> E </Object> </Object>
</Object> </Dependency>
```

(a) Input XML

Parent	Child
A	B
A	C
B	D
D	F
D	G

(b) Output relation

$$P \coloneqq \lambda\tau.\, filter(\psi, \lambda t, \phi_1 \wedge \phi_2)$$

$$\psi \coloneqq (\lambda s.\, \pi_1)\{root(\tau)\} \times (\lambda s.\, \pi_2)\{root(\tau)\}$$

$$\pi_1 = \pi_2 = pchildren(descendants(s, Object), text, 0)$$

$$\phi_1 : \Big((\lambda \mathrm{n}.\, \mathrm{child}(\mathrm{parent}(\mathrm{n}), \mathrm{id}, 0))\Big)\, \mathrm{t}[0] < 20$$

$$\phi_2 : (\lambda \mathrm{n}.\, \mathrm{parent}(\mathrm{n}))\mathrm{t}[0] = (\lambda \mathrm{n}.\, \mathrm{parent}(\mathrm{parent}(\mathrm{n}))\mathrm{t}[1])$$

(c) Synthesized program

Figure 7. Synthesis program for example input and output and example 3
图 7. 输入输出示例和示例 3 的综合程序

示例 3：考虑图 7 中所示的数据转换任务，其目的是将 XML 文件中每个 id 小于 20 的对象元素的文本值映射到其直接嵌套对象元素的文本值。图 7(c)显示了这种转换的 DSL 程序。在这里，列提取器 π_1、π_2 使用提取函数 *descendants* 和 *pchildren* 来提取所有带有标签文本的子节点，并且可以从根节点到达任何对象节点的 *pos* 0。谓词 ϕ_1 过滤掉所有元组中第一个元素的 id 大于或等于 20 的元组。第二个谓词 ϕ_2 确保元组中的第二个元素直接嵌套在第一个元素内。

3.3. 合成算法

本节中介绍用于将层次树从输入输出示例转换为关系表的合成算法。通过对目标数据库中的每个表执行一次算法，本文的方法可用于将 XML 或 JSON 文档转换为关系表。

合成算法的顶层结构在算法 1 中给出。该算法采用一组形式为$\{T_1 \to R_1, \cdots, T_m \to R_m\}$的输入—输出示例，其中每个 T_i 代表一个层次树(输入示例)和 R_i 表示其所需的关系表表示(输出示例)。

由于目标表的模式在实际中通常是固定的，本文假设所有输出表(即 R_i)具有相同数量的列。给定这些输入输出示例，学习转换过程(Learn Transformation)在本文的 DSL 中返回一个与所有输入输出示例一致的程序 P^*外，由于 P^*是满足规范的最简单的 DSL 程序，因此预设它是一个不会过拟合示例的通用程序。

本文的方法将合任务分解为提取列和行的两个阶段。在第一阶段，合成了一个列提取程序，该程序提取的列与输出表 R 中的每一列过度近似。通过这些列提取程序得到的列的笛卡尔积进一步产生了一个

过度近似目标表 R 的表 R'。第二阶段，本文学习了一个谓词 ϕ，过滤掉 R'中没有出现在输出表 R 中的“虚假”元组。

算法 1 总体合成算法

1: 学习转换(ε)
2: **Input**：输入输出示例 $\varepsilon=\{T_1\to R_1,\cdots,T_m\to R_m\}$。
//一组层次树及其所需的关系表作为输入输出示例
3: **Output**：最简单的领域特定语言程序 P^*。
4: **for all** $1\le j\le k$ **do**
5: Π_j := 列学习提取(ε, j)；
6: $\Psi:=\Pi_1\times\cdots\times\Pi_n$；
//调用列提取过程，取提取列的笛卡尔积作为表提取做准备
7: $P^*:=\perp$；
8: **for all** $\psi\in\Psi$ **do**
9: ϕ:= 学习谓词(ε, ψ)；
10: **if** $\phi\neq\perp$ **then**
11: $P:=\lambda\tau.filter(\psi,\lambda t.\phi)$
12: **if** $\theta(P)<\theta(P^*)$ **then** $P^*:=P$
//学习谓词 ϕ 并使用，过滤掉上一过程表提取程序集合中的虚假元组
13: **return** P^*；

现在介绍更详细地算法 1 中的学习转换过程(Learn Transformation)。总体合成算法首先调用列学习提取过程(Learn Col Extractors)，该过程学习一组列提取表达式 Π_j，这样在每个 T_i 上应用任何 $\pi\in\Pi_j$ 都会产生产生一列，该列高度近似表 R_i 中的第 j 列(第 4~5 行)。

本文算法基于确定性有限自动机的列学习提取过程，将在第 3.3.1 节中更详细地描述。

当每列有了单独的列提取程序后，就可以通过获取每个列提取程序的笛卡尔积来获得所有可能的表提取程序的集合(第 6 行)。因此，将每个表提取程序 $\psi\in\Psi$ 应用于输入树 T_i 会产生一个表 R_i'，它高度近似输出表 R_i。

合成算法的下一步(第 7~12 行)学习过滤器构造中使用的谓词。对于每个表提取程序 $\psi\in\Psi$，本文使用学习一个谓词 ϕ，使得 $\lambda\tau.filter(\psi,\lambda t.\phi)$ 与示例一致。具体来说，学习谓词过程(Learn Predicate)产生一个谓词，过滤掉 $[\![\psi]\!]_{T_i}$ 中的虚假元组。

如果没有这样的谓词，学习谓词过程(Learn Predicate)返回⊥。本文的谓词学习算法使用整数线性规划来推断具有最少原子谓词数量的最简单公式。本文在第 3.3.2 节中更详细地描述了 Learn Predicate 算法。

由于可能有多个 DSL 程序满足提供的示例，本文的方法使用奥卡姆剃刀原理在不同的解决方案之间进行选择。特别是，算法 1 使用启发式成本函数 θ 来确定哪个解决方案更简单(第 12 行)，因此保证返回一个最小化 θ 值的程序。即 θ 为使用复谓词或列提取器的程序分配了更高的成本。

3.3.1. 列学习提取程序

本文用于列学习提取程序的方法基于确定性有限自动机(Deterministic Fifinite Automata)。给定一棵树(输入示例)和单列(输出示例)，本文的方法构造了一个 DFA，其语言完全接受与输入－输出示例一致的列提取程序集。如果有多个输入输出示例，则为每个示例构建一个单独的 DFA，然后取它们的交集。生成 DFA 的语言包括与所有示例一致的列提取程序。

算法 2 列学习提取的算法

1: 列学习提取(ε, i)
2: **Input：** 一组示例 ε 和要学习提取的列号 i。
3: **Output：** Set Π of column extractors。
//返回一组列提取程序 Π 使对于每个 $\pi \in \Pi$ 和
每个输入－输出示例(T, R)有 $[\![\pi]\!]_{\{T.root\},T} \supseteq column(R,i)$
4: $\varepsilon' := \{(T,\kappa) \mid (T,R) \in \varepsilon \wedge \kappa = column(R,i)\}$
5: A:= 构建 DFA(e_0) where $e_0 \in \varepsilon'$
6: **for all** $e \in \varepsilon' \setminus \{e_0\}$ **do**
7: A':= 构建 DFA(e)
8: A:= INTERSECT (A, A')
9: **return** LANGUAGE (A)

$$\frac{s=\{T.root\}}{q_0=q_s \in Q} \quad (1)$$

$$\frac{q_s \in Q \ tag \ is \ a \ tag \ in \ T \quad [\![children\ (s,tag)]\!]_{s,T}=s'}{q_{s'} \in Q, \delta(q_s, children_{tag})=q_{s'}} \quad (2)$$

$$\frac{q_s \in Q \ tag \ is \ a \ tag \ in \ T \quad pos \ is \ a \ pos \ in \ T \quad [\![pchildren\ (s,tag,pos)]\!]_{s,T}=s'}{q_{s'} \in Q, \delta(q_s, pchildren_{tag,pos})=q_{s'}} \quad (3)$$

$$\frac{q_s \in Q \ tag \ is \ a \ tag \ in \ T \quad [\![descendants\ (s,tag)]\!]_{s,T}=s'}{q_{s'} \in Q, \delta(q_s, descendants_{tag})=q_{s'}} \quad (4)$$

$$\frac{s \supseteq column\ (R,i)}{q_s \in F} \quad (5)$$

Figure 8. FTA construction rules. T is the input tree, R is the output table, and i is the column to extract
图 8. FTA 构建规则。T 是输入树，R 是输出表，i 是要提取的列

算法 2 首先构建一组示例 ε'，将每个输入树映射到输出表的第 i 列(第 4 行)。然后，对于每个示例 $e \in \varepsilon'$，本文构建一个确定性有限自动机，代表所有与 e 一致的列提取程序(第 5~8 行)。与所有示例 ε'一致的程序集对应于第 9 行的交叉自动机 A 的语言。特别是，第 8 行中使用的 Intersect 程序是 DFA [29]的标准交叉程序。具体来说，两个 DFA，A_1 和 A_2 的交集只接受 A_1 和 A_2 都接受的程序，并通过取 A_1 和 A_2 的笛卡尔积并将接受状态定义为形式(q_1, q_2)，其中 q_1 和 q_2 分别接受 A_1 和 A_2 的状态。

列学习提取过程的关键部分是构建 DFA 方法，它使用图 8 所示的规则从输入输出示例构建确定性有限自动机 $A=(Q,\Sigma,\delta,q_0,F)$。自动机的状态 Q 对应于输入 HDT 中的节点集。使用符号 q_s 来表示表示节点集 s 的状态。字母 Σ 对应于 DSL 中列提取函数的名称，具体来说，

本文有：Σ = {$children_{tag}$|tag 是 T 中的一个标签}
∪ {$pchildren_{tag,\ pos}$|tag(pos)是 T 中的一个标签(位置)}
∪ {$descendants_{tag}$|tag 是 T 中的一个标签}

字母表中的每个符号对应一个领域特定语言操作符(使用输入层次树的标签和位置进行实例化)。DFA 中的转换是使用领域特定语言运算符的语义构造的，给定领域特定语言构造 $f \in \{children, pchildren, descendants\}$ 和状态 q_s，如果将 f 应用于 s 产生 s'，则 DFA 包含转换 $q_s \xrightarrow{f} q'_s$。

DFA 的初始状态是{*T.root*}，本文有 $q_s \in F$ 如果 *s* 高度近似表 *R* 中的第 *i* 列。

图 8 中所示的构造规则。其中规则(1)~(4)处理本文 DSL 中的列提取器构造并构造状态和/或转换。第一条规则添加 *q*{*T.root*}作为初始状态，因为层次树的根节点是直接可达的。第二条规则添加状态 q_s' 和转换 $\delta(q_s, childrentag) = q_s'$ 如果 *children*(*s*, *tag*)的计算结果为 *s*′。规则(3)和(4)与规则(2)类似，处理剩余的列提取函数 *pchildren* 和 *descendants*。例如，如果 *pchildren*(*s*, *tag*, *pos*)的计算结果为 *s*′，则本文有 $\delta(q_s, pchildrentag, pos) = q_s'$。图 8 中的最后一条规则标识了最终状态。特别地，如果 *s* 是目标列(即输出示例)的超集，则状态 q_s 是最终状态。

定理 1：令 A 为算法 2 为一组输入输出示例 ε 和列号 *i* 构造的 DFA。然后，A 在本文的领域特定语言中接受列提取程序 π，iff $\forall(T,R) \in \varepsilon$。$[\![\pi]\!]_{\{T.root\},T} \supseteq column(R,i)$。

示例 4：假设使用图 8 中的规则构建的 DFA 接受单词 *xy*，其中 $x = descendants_{object}$ 和 $y = pchildren_{text,0}$。这个词对应于以下列提取程序：

$$\pi = pchildren(descendants(s, object), text, 0)$$

如果 T 是输入示例，而 *column*(*R*, *i*)是输出示例，那么本文有 $((\lambda s.\pi)\{T.root\}) \supseteq column(R,i)$。

3.3.2. 学习谓词

该算法首先构建可用于公式 ϕ(第 4 步)的所有可能原子谓词的(有限) Φ。这些谓词是使用图 9 中的规则为一组输入－输出示例 ε 和候选表提取器 ψ 构建的。其中 x_i 表示一组节点提取器，可以应用于为第 *i* 列提取的节点。根据这个规则，生成一个最小谓词 $((\lambda n.\varphi)t[i] \trianglelefteq c)$。如果 *i* 是范围 $[1,k]$ 中的有效列号，*c* 是输入表之一中的常量，并且 φ 是第 *i* 列的有效节点提取器。图 9 中的规则(1)~(3)定义了节点提取器 φ 对列 *i* 有效的含义，表示为 $\varphi \in x_i$。如果对生成的中间表的第 *i* 列中的任何条目应用 φ 不会“引发异常”(即产生⊥)，可以称 φ 是第 *i* 列的有效节点提取器。

算法 3 学习谓词算法

1: 学习谓词(ε, ψ)
2: **Input**：示例 ε，一个候选表提取器 ψ。
3: **Output**：所需的谓词 ϕ。
//返回谓词 ϕ，使得对于每个 $(T,R) \in \varepsilon$，程序 $filter(\psi, \lambda t.\phi)$ 在输入 *T* 上产生所需的输出表 *R*
4: Φ := 构建谓词库(ε, ψ)
5: $\varepsilon^+ := \varnothing$； $\varepsilon^- := \varnothing$
6: **for all** $(T,R) \in \varepsilon$ **do**
7: **for all** $t \in [\![\psi]\!]_T$ **do**
8: **if** $t \in R$ **then**
9: $\varepsilon^+ := \varepsilon^+ \cup \{t\}$
10: **else** $\varepsilon^- := \varepsilon^- \cup \{t\}$
11: Φ^* := 查找最小谓词(Φ, ε^+, ε^-)
12: 查找满足以下条件的 ϕ：
13: (1) ϕ 是 Φ^*中谓词的布尔组合
14: (2) $\phi(t) = \begin{cases} 1 & \text{if } t \in \varepsilon^+ \\ 0 & \text{if } t \in \varepsilon^- \end{cases}$
15: **return** ϕ

$$\frac{}{\pi_i, \varepsilon \vdash n \in x_i} \quad (1)$$

$$\frac{\pi_i,\varepsilon\vdash\varphi\in x_i \quad \forall_T\to R\in\varepsilon.\forall_n\in\pi_i(T).[\![parent\ (\varphi)]\!]_{n,T}\neq\perp}{\pi_i,\varepsilon\vdash parent\ (\varphi)\in x_i} \quad (2)$$

$$\frac{\pi_i,\varepsilon\vdash\varphi\in x_i \quad \forall_T\to R\in\varepsilon.\forall_n\in\pi_i(T).[\![child\,(\varphi,tag,pos)]\!]_{n,T}\neq\perp}{\pi_i,\varepsilon\vdash child\,(\varphi,tag,pos)\in x_i} \quad (3)$$

$$\frac{\psi=\pi_1\times\cdots\times\pi_k,\quad i\in[1,k] \quad \pi_i,\varepsilon\vdash\varphi\in x_i \quad \exists(T,R)\in\varepsilon.c\in data\ (T)}{\psi,\varepsilon\vdash((\lambda n.\varphi)\ t[i])\trianglelefteq c\in\Phi} \quad (4)$$

$$\frac{\psi=\pi_1\times\cdots\times\pi_k,\quad i\in[1,k],j\in[1,k] \quad \pi_i,\varepsilon\vdash\varphi_1\in x_i \quad \pi_j,\varepsilon\vdash\varphi_2\in x_j}{\psi,\varepsilon\vdash((\lambda n.\varphi_1)\ t[i])\trianglelefteq((\lambda n.\varphi_2)\ t[j])\in\Phi} \quad (5)$$

Figure 9. Predicate library construction rules
图 9. 谓词库构建规则

算法 3 的第 5 步到第 10 步是构建一组正例和负例用于学习谓词。正例指的是应该出现在所需输出表中的元组，负例对应应该被过滤掉的元组。谓词学习器的目标是在谓词库 Φ 中的原子谓词上找到一个，使得对所有正例评估为真，对所有负例评估为假。即充当和之间的分类器。

为了学习合适的分类器，本文的算法首先学习了区分 ε^+ 样本和 ε^- 样本所必需的最小谓词集。由于本文的目标是合成一个不会过拟合输入输出示例的通用程序，因此需要合成谓词 ϕ 尽可能简单。本文将寻找最简单分类器的问题表述为整数线性规划(ILP)和逻辑最小化[30] [31]的组合。使用 ILP 来学习必须在分类器中使用的最小谓词 Φ^*集合，然后使用逻辑最小化技术 Φ^*在上找到具有最少布尔连接词数量的。

本文查找最小谓词集的方法在算法 4 中给出。通过以下方式将其减少到 0~1 ILP 来解决这个优化问题：首先引入一个指示变量 x_k，如果 $p_k \in \Phi$ 被选择在 Φ^*中，则 $x_k = 1$，否则 $x_k = 0$。然后，对于每个谓词 p_k 和每对示例$(e_i, e_j \in \varepsilon^+ \times \varepsilon^-)$，引入一个常数变量 a_{ijk}，如果谓词 p_k 区分 e_i 和 e_j，我们将 a_{ijk} 赋值为 1，否则赋值为 0。观察到每个 a_{ijk} 的值是已知的，而对每个变量 x_k 的赋值是有待确定的。

算法 4 查找最小谓词集算法

1: 查找最小谓词(Φ, ε^+, ε^-)
2: **Input：** 谓词库全域 Φ
3: **Input：** 正例 ε^+，负例 ε^-
4: **Output：** 谓词集 Φ^*满足 $\Phi^* \subseteq \Phi$
//返回 Φ 的子集 Φ^*使得，对于每对示例 $(e_1, e_2) \in \varepsilon^+ \times \varepsilon^-$，存在一个原子谓词 $p \in \Phi^*$ 使得 p 对 e_1 和 e_2 求得不同的真值，以区分 e_1 和 e_2
5: **for all** $(\phi_k, e_i, e_j) \in \Phi \times \varepsilon^+ \times \varepsilon^-$ **do**
6: $a_{i,j,k} = \begin{cases} 1 \text{ if } \phi_k(e_i) \neq \phi_k(e_j) \\ 0 \quad \text{otherwise} \end{cases}$
7: minimize $\sum_{k=1}^{|\Phi|} x_k$
8: subject to:
9: $\forall (e_i, e_j) \in \varepsilon^+ \times \varepsilon^- . \sum_{k=1}^{|\Phi|} a_{ijk} x_k \geq 1$
10: $\wedge \quad \forall k \in [1, |\Phi|] . x_k \in \{0,1\}$
11: **return** $\{\phi_i \mid \phi_i \in \Phi \wedge x_i = 1\}$

回到算法 3，查找最小谓词过程(第 11 行)的返回值 Φ^*对应于必须在分类器中使用的最小谓词集；然而，本文仍然需要在 Φ^*中找到谓词的布尔组合，以区分 ε^+ 样本和 ε^- 样本。在 Φ^*中的谓词上找到最小分类器的问题作为逻辑最小化问题[30] [32]。给定一组谓词 Φ^*，本文的目标是在 Φ^*中的谓词上找到最小的 DNF 公式 ϕ，使得 ϕ 对于任何正例评估为真，对于任何负例评估为假。为了解决这个问题，本文首先构建一个(部分)真值表，其中行对应于 $\varepsilon^+ \bigcup \varepsilon^-$ 中的示例，列对应于 Φ^*中的谓词。如果谓词 $p_j \in \Phi^*$ 为真，例如 e_i，则真值表的第 i 行和第 j 列中的条目为真，否则为假。目标布尔函数 ϕ 应该对任何 $e^+ \in \varepsilon^+$ 评估为真，对任何 $e^- \in \varepsilon^-$ 评估为假。由于本文有一个描述目标布尔函数的真值表，本文可以使用标准技术，例如 Quine-McCluskey 方法[30] [32]，找到表示分类器 ϕ 的最小 DNF 公式。

定理 2：给定示例 ε 和表提取器 ψ，使得 $\forall (T,R) \in \varepsilon$， $R \subseteq [\![\psi]\!]_T$，如果本文的 DSL 中存在这样的公式 $[\![filter(\psi, \lambda t.\phi)]\!]_T = R$，使得 $\forall (T,R) \in \varepsilon$，算法 3 返回最小 DNF 公式 ϕ。

例 5：考虑谓词域 $\Phi = \{\phi_1, \cdots, \phi_7\}$，一组正例 $\varepsilon^+ = \{e_1^+, e_2^+, e_3^+\}$，和一组负例 $\varepsilon^- = \{e_1^-, e_2^-, e_3^-\}$ 与图 7(a)中给出的真值表。此处，图 7(a)的第 e_i 行和第 ϕ_j 列的条目为真，只要元组 e_i 满足谓词 ϕ_j。谓词学习器的目标是找到一个公式 $\phi_?$ 具有最少数量的原子谓词 $\phi_j \in \Phi$，使得它对所有正例评估为真，对所有负例评估为假。为此，查找最小谓词过程首先找到所需的最小谓词 Φ^*子集，如算法 4 中所述。

	ϕ_1	ϕ_2	ϕ_3	ϕ_4	ϕ_5	ϕ_6	ϕ_7
e_1^+	*true*	*true*	*false*	*false*	*true*	*true*	*false*
e_2^+	*false*	*true*	*true*	*true*	*true*	*false*	*true*
e_3^+	*false*	*true*	*true*	*true*	*false*	*false*	*false*
e_1^-	*false*	*false*	*true*	*true*	*false*	*false*	*false*
e_2^-	*false*	*true*	*true*	*true*	*false*	*false*	*true*
e_3^-	*true*	*false*	*true*	*false*	*false*	*false*	*true*

(a) 谓词全域 Φ、正例 ε^+ 和负例 ε^- 的初始真值表

	v_{11}	v_{12}	v_{13}	v_{21}	v_{22}	v_{23}	v_{31}	v_{32}	v_{33}
ϕ_1	1	1	0	0	0	1	0	0	1
ϕ_2	1	0	1	1	0	1	1	0	1
ϕ_3	1	1	1	0	0	0	0	0	0
ϕ_4	1	1	0	0	0	1	0	0	1
ϕ_5	1	1	1	1	1	1	0	0	0
ϕ_6	1	1	1	0	0	0	0	0	0
ϕ_7	0	1	1	1	0	0	0	1	1

(b) 在算法 4 中分配的 a_{ijk} 的值。这里 v_{ij} 对应于 $(e_i, e_j) \in \varepsilon^+ \times \varepsilon^-$

	ϕ_2	ϕ_5	ϕ_7	$\phi_?$
e_1^+	*true*	*true*	*false*	*true*
e_2^+	*true*	*true*	*true*	*true*
e_3^+	*true*	*false*	*false*	*true*
e_1^-	*false*	*false*	*false*	*false*
e_2^-	*true*	*false*	*true*	*false*
e_3^-	*false*	*false*	*true*	*false*

(c) 算法 3 构建的真值表

Figure 10. Truth table and values for Example 5
图 10. 示例 5 的 a_{ijk} 真值表和值

图 10 显示了算法 4 第 6 行中分配的 a_{ijk} 值。特别是，图 10 的行 ϕ_i 和列 v_{jk} 处的条目为真，当条件是 a_{ijk} 为真。图 10(b)中突出显示的行表示使用整数线性规划选择在 Φ^*中的谓词。找到 $\Phi^* = \{\phi_2, \phi_5, \phi_7\}$ 后，第 12~14 行算法 3 生成(部分)真值表，如图 10(c)所示。与这个真值表一致的最小 DNF 公式是 $\phi_5 \vee (\phi_2 \wedge \neg\phi_7)$，所以本文的算法返回这个公式作为分类器。

4. 实验

4.1. 实验目的

本文进行了两组实验，旨在分别验证合成算法实现层次模型到关系模型数据转换的效果，以及完成现实数据集转换任务的效率。第一个实验在从某地产收集的已经完成树到表转换任务的数据集上评估本文算法的精度和运行时间，第二个实验是使用本文的方法将现实城市地产的层次模型数据集转换为关系模型，实现数据模式转换。

4.2. 实验数据集

本文使用了某城市房地产的相关数据集，也是本文示例中数据集的来源。包含某城市大约四十万条的房地产信息，第一个实验采用的是从相关网站抓取数据后，已经完成层次结构到关系表转换的数据集，包含了具有统一目标结构的楼盘表、楼栋表、户表、许可表、等表结构的数据。第二个实验使用了某地产两个需要转换为目标关系模型的现实城市地产数据集。两个实验使用的数据都包含了 XML 和 JSON 文件。

4.3. 精度和运行时间

在评估算法精度和运行时间实验中，使用了 38 个(21 个 XML 文件，17 个 JSON 文件)某地产收集的已经完成树到表转换任务的数据集。直接从 38 个基准测试中获取的输入 XML/JSON 文件作为输入示例。对于输出示例，一部分使用某地产提供的表格，另一部分采用构建所需输出表的方式。对于每个基准，使用算法合成一个执行给定任务的程序。手动检查了输出，以查看合成程序是否执行所需的功能。

Table 1. Summary of trial evaluations
表 1. 试验评估总结

		准确性		综合时间	
	列数	基准数据集	正确解决的基准数量	中位数(秒)	平均数(秒)
XML	≤2	5	4	0.35	0.39
	3	6	6	0.65	3.57
	4	5	4	1.27	3.59
	≥5	5	5	3.49	6.82
	合计	**21**	**19**	**0.84**	**3.32**
JSON	≤2	4	4	0.12	0.25
	3	3	3	0.46	1.15
	4	4	4	0.29	11.20
	≥5	6	5	3.21	3.88
	合计	**17**	**16**	**0.33**	**4.43**
总计		**38**	**35**	**0.54**	**3.77**

表 1 总结了在这 38 个基准上评估算法的结果。该表的第一部分提供了有关基准数据集的信息，分别是按照列数划分的每个类别中基准数，能够正确解决的基准数量。本文的方法能够为 92.1%的这些基准测试同步目标程序。

“合成时间”下的列显示合成所需程序所需的中位数和平均时间(以秒为单位)。取平均数时，在 3.8 秒内合成目标转换，中位时间为 0.5 秒。

4.4. 实现关系模式转换

在这个实验中，首先对于每个目标表，提供一对输入输出示例，以及关系模型中所有主键和外键的列表。然后检查合成程序并验证其正确性。该实验的结果总结在表 2 中。

Table 2. Hierarchical model real estate dataset to relational model data conversion
表 2. 层次模型真实地产数据集到关系模型数据转换

名称	数据集		数据库		合成		实验执行	
	格式	大小	表数	列数	总时间(s)	平均时间(s)	总时间(s)	平均时间(s)
楼盘数据集 1	XML	0.97 GB	5	27	4.57	0.91	545.53	109.10
楼盘数据集 2	JSON	4.22 GB	5	25	13.61	2.72	643.11	128.62

“数据库”下的列显示了目标关系模型中的表数和属性总数。“合成”下的两列分别显示总合成时间和平均合成时间。平均时间表示每个表的合成时间，总时间是合成所有数据库表时间的和。如表 2 所示，每个数据库表的平均合成时间在 1.2 到 2.7 秒之间。

最后一部分显示合成程序在原始完整数据集上的执行时间的统计信息。这些数据表明，本文的方法可用于实现真实数据集的层次模型到关系模型的数据转换。

5. 总结

本文提出了一种基于示例编程的方法，用于将 XML 和 JSON 文档等层次模型数据转换为关系模型。方法的核心思想是将合成任务分解为两个阶段，分别进行列学习和行学习。通过给定一个小的输入输出示例，算法合成所需的数据转换程序。根据收集的现实 XML 和 JSON 数据集，并经过实验验证，本文提出的方法能够为现实世界的层次结构数据合成所需的程序，实现到关系模型的数据转换。测试结果符合精度和运行时间的要求，并在很少的用户操作下完成将 XML 文档或 JSON 文档转换为关系数据表。不仅便于高效的数据提取，满足用户进行存储和查询的需求，而且减少了转换设计工作，减少了人力消耗。

参考文献

[1] Fagin, R., Kolaitis, P.G., Miller, R.J. and Popa, L. (2005) Data Exchange: Semantics and Query Answering. *Theoretical Computer Science*, **336**, 89-124. https://doi.org/10.1016/j.tcs.2004.10.033

[2] Popa, L., Velegrakis, Y., Hernandez, M.A., Miller, R.J. and Fagin, R. (2002) Translating Web Data. *Proceedings of the* 28*th International Conference on very Large Data Bases*, Hong Kong, 20-23 August 2002, 598-609. https://doi.org/10.1016/B978-155860869-6/50059-7

[3] Fagin, R., Haas, L.M., Hernandez, M., Miller, R.J., Popa, L. and Velegrakis, Y. (2009) Clio: Schema Mapping Creation and Data Exchange. In: *Conceptual Modeling*: *Foundations and Applications*, Springer, Berlin, 198-236. https://doi.org/10.1007/978-3-642-02463-4_12

[4] Roth, M. and Tan, W.-C. (2013) Data Integration and Data Exchange: It’s Really about Time. *Sixth Biennial Conference on Innovative Data Systems Research*, *CIDR* 2013, Asilomar, 6-9 January 2013.

[5] Miller, R.J., Haas, L.M. and Hernandez, M.A. (2000) Schema Mapping as Query Discovery. *VLDB* 2000, *Proceedings of* 26*th International Conference on very Large Data Bases*, Cairo, 10-14 September 2000, Volume 2000, 77-88.

[6] Fagin, R., Kolaitis, P.G. and Popa, L. (2005) Data Exchange: Getting to the Core. *ACM Transactions on Database Systems* (*TODS*), **30**, 174-210. https://doi.org/10.1145/1061318.1061323

[7] Kolaitis, P.G. (2005) Schema Mappings, Data Exchange, and Metadata Management. *Proceedings of the Twenty-Fourth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Baltimore, 13-15 June 2005, 61-75. https://doi.org/10.1145/1065167.1065176

[8] Kang, J. and Naughton, J.F. (2003) On Schema Matching with Opaque Column Names and Data Values. *Proceedings of the* 2003 *ACM SIGMOD International Conference on Management of Data*, San Diego, 9-12 June 2003, 205-216. https://doi.org/10.1145/872757.872783

[9] Madhavan, J., Bernstein, P.A. and Rahm, E. (2001) Generic Schema Matching with Cupid. *VLDB* 2001, *Proceedings of* 27*th International Conference on Very Large Data Bases*, Roma, 11-14 September 2001, 49-58.

[10] Do, H.-H. and Rahm, E. (2002) COMA: A System for Flexible Combination of Schema Matching Approaches. *Proceedings of* 28*th International Conference on Very Large Data Bases*, *VLDB* 2002, Hong Kong, 20-23 August 2002, 610-621. https://doi.org/10.1016/B978-155860869-6/50060-3

[11] Nandi, A. and Bernstein, P.A. (2009) HAMSTER: Using Search Clicklogs for Schema and Taxonomy Matching. *PVLDB*, **2**, 181-192. https://doi.org/10.14778/1687627.1687649

[12] Elmeleegy, H., Ouzzani, M. and Elmagarmid, A. (2008) Usage-Based Schema Matching. *Proceedings of the* 24*th International Conference on Data Engineering Workshops*, *ICDE* 2008, Cancún, 7-12 April 2008, 20-29. https://doi.org/10.1109/ICDE.2008.4497410

[13] Qian, L., Cafarella, M.J. and Jagadish, H. (2012) Sample-Driven Schema Mapping. In: *Proceedings of the* 2012 *ACM SIGMOD International Conference on Management of Data*, ACM, New York, 73-84. https://doi.org/10.1145/2213836.2213846

[14] Yan, L.L., Miller, R.J., Haas, L.M. and Fagin, R. (2001) Data-Driven Understanding and Refinement of Schema Mappings. *ACM SIGMOD*, Volume 30, 485-496. https://doi.org/10.1145/376284.375729

[15] Alexe, B., Chiticariu, L., Miller, R.J. and Tan, W.-C. (2008) Muse: Mapping Understanding and Design by Example. *Proceedings of the* 24*th International Conference on Data Engineering Workshops*, *ICDE* 2008, Cancún, 7-12 April 2008, 10-19. https://doi.org/10.1109/ICDE.2008.4497409

[16] Krishnamurthy, R. (2004) Xml-to-sql Query Translation. PhD Thesis, University of Wisconsin, Madison. https://doi.org/10.1016/B978-012088469-8.50016-4

[17] Shanmugasundaram, J., Shekita, E., Kiernan, J., Krishnamurthy, R., Viglas, E., Naughton, J. and Tatarinov, I. (2001) A General Technique for Querying Xml Documents Using a Relational Database System. *ACM SIGMOD Record*, **30**, 20-26. https://doi.org/10.1145/603867.603871

[18] Dweib, I., Awadi, A., Elrhman, S.E.F. and Lu, J. (2008) Schemaless Approach of Mapping XML Document into Relational Database. 8*th IEEE International Conference on Computer and Information Technology*, Sydney, 8-11 July 2008, 167-172. https://doi.org/10.1109/CIT.2008.4594668

[19] Yoshikawa, M., Amagasa, T., Shimura, T. and Uemura, S. (2001) XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. *ACM Transactions on Internet Technology*, **1**, 110-141. https://doi.org/10.1145/383034.383038

[20] Jiang, H., Lu, H., Wang, W. and Yu, J.X. (2002) XParent: An Efficient RDBMS-Based XML Database System. *Proceedings of the* 18*th International Conference on Data Engineering*, San Jose, 26 February-1 March 2002, 335-336.

[21] Tatarinov, I., Viglas, S.D., Beyer, K., Shanmugasundaram, J., Shekita, E. and Zhang, C. (2002) Storing and Querying Ordered XML Using a Relational Database System. *The* 2002 *ACM SIGMOD International Conference on Management of Data*, Wisconsin, 4-6 June 2002, 204-215. https://doi.org/10.1145/564691.564715

[22] Shanmugasundaram, J., Tufte, K., Zhang, C., He, G., DeWitt, D.J. and Naughton, J.F. (1999) Relational Databases for Querying XML Documents: Limitations and Opportunities. *Proceedings of* 25*th International Conference on Very Large Data Bases*, Edinburgh, 7-10 September 1999, 302-314.

[23] Amer-Yahia, S., Du, F. and Freire, J. (2004) A Comprehensive Solution to the XML-to-Relational Mapping Problem. In: *Proceedings of the* 6*th Annual ACM International Workshop on Web Information and Data Management*, ACM, New York, 31-38. https://doi.org/10.1145/1031453.1031461

[24] Atay, M., Chebotko, A., Liu, D., Lu, S. and Fotouhi, F. (2007) Efficient Schema-Based XML-to-Relational Data Mapping. *Information Systems*, **32**, 458-476. https://doi.org/10.1016/j.is.2005.12.008

[25] Soltan, S. and Rahgozar, M. (2006) A Clustering-Based Scheme for Labeling XML Trees. *International Journal of Computer Science and Network Security*, **6**, 84-89.

[26] Fujimoto, K., Kha, D.D., Yoshikawa, M. and Amagasa, T. (2005) A Mapping Scheme of XML Documents into Relational Databases Using Schema-Based Path Identifiers. *International Workshop on Challenges in Web Information Retrieval and Integration*, Tokyo, 8-9 April 2005, 82-90.

[27] Xing, G., Xia, Z. and Ayers, D. (2007) X2R: A System for Managing XML Documents and Key Constraints Using RDBMS. *ACMSE* 2007, Winston-Salem, 23-24 March 2007, 215-220. https://doi.org/10.1145/1233341.1233380

[28] Imdb to json. https://github.com/oxplot/imdb2json/blob/master/Readme.md

[29] Hopcroft, J.E., Motwani, R. and Ullman, J.D. (2006) Introduction to Automata Theory, Languages, and Computation. 3rd Edition, Addison-Wesley Longman Publishing Co., Inc., Boston.

[30] McCluskey, E.J. (1956) Minimization of Boolean Functions. *Bell Labs Technical Journal*, **35**, 1417-1444. https://doi.org/10.1002/j.1538-7305.1956.tb03835.x

[31] Quine, W.V. (1952) The Problem of Simplifying Truth Functions. *The American Mathematical Monthly*, **59**, 521-531. https://doi.org/10.1080/00029890.1952.11988183

[32] Madhavan, J., Bernstein, P.A., and Rahm, E. (2001) Generic Schema Matching with Cupid. VLDB, 49-58.