THE NATIONAL UNIVERSITY
of SINGAPORE

School of Computing
Computing 1, 13 Computing Drive, Singapore 117417

TRA3/12

Discovering Semantics from XML

Luochen Li, Thuy Ngoc Le, Tok Wang Ling, Huayu Wu and Stephane Bressan.

March 2012

T e c h n i c a l R e p o r t

Foreword

This technical report contains a research paper, development or tutorial article, which has been submitted for publication in a journal or for consideration by the commissioning organization. The report represents the ideas of its author, and should not be taken as the official views of the School or the University. Any discussion of the content of the report should be sent to the author, at the address shown on the cover.

OOI Beng Chin
Dean of School

Discovering Semantics from XML

Luochen Li [#], Thuy Ngoc Le [#], Tok Wang Ling [#],
Huayu Wu [*], and Stephane Bressan [#]

[#]School of Computing, National University of Singapore
{luochen,ltngoc,lingtw,steph}@comp.nus.edu.sg
[*] Institute for Infocomm Research, Singapore
huwu@i2r.a-star.edu.sg

Abstract. In database applications in general, and in applications using XML in particular, the availability of a conceptual schema or of elements of semantics constitute invaluable leverage for improving the effectiveness, and sometimes the efficiency, of many tasks including query processing, keyword search and schema and data integration. The Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) model is a conceptual model designed to capture the semantics of the important constructs in a variety of data models in general and in semi-structured data models in particular. It is specifically intended to capture the semantics of object classes, object identifiers, relationship types, object attributes and relationship attributes underlying XML schemas and data. For a given application, we refer to the set of instances of these semantic concepts as the ORA-semantics of the application. While ORA-SS can be used a priori for the design of new applications, we are interested, in this work, in the automatically discovering of the ORA-semantics from existing XML data and XML schemas. In this paper, we present a novel approach to automatically discover the ORA-semantics from XML schemas and XML data. The approach we proposed is based on a set of techniques and heuristics that identify the different semantic concepts. We empirically and comparatively evaluate the effectiveness of the approach.

1 Introduction

Actors in several industry sectors such as the automotive industry and the chemical industry are now commonly using XML to define standards, in the form of XML schemas, for the representation of technical and business data, and to electronically interchange data among business partners. Indeed, XML can effectively replace archaic electronic data interchange formats, which were confusing physical, logical and conceptual levels in the name of efficiency,and which were generally hard to understand by users, thus creating inflexible information systems. Orthogonal techniques such as compression can provide the wired efficiency. The conceptual quality is provided by the logical nature of the XML data model and the suite of tools it techniques that accompany it such as query languages, keyword search and techniques for schema and data integration.

However,in order to improve the conceptual quality, one needs to discover the intended semantics in the logical XML schemas and data. This requires finding such semantic information as object classes, relationship types, object identifiers, object and relationship attributes, as present in conceptual models for semi-structured data such Object-Relationship-Attribute for Semi-Structured data (ORA-SS) [10]. We refer to this implicit semantics as the ORA-semantics.

Once discovered, the ORA-semantics is useful not only for users to understand the data and schemas but also for improving both the effectiveness and efficiency of processing. We use the XML document in Fig. 1 and the examples to illustrate how the availability of such semantics positively impacts applications.

XML query processing. To process an XPath query, e.g., *//Student[Matric# ="HT001"]/Name* to the document in Fig. 1, most query engines will match the query pattern to the document to find all occurrences. However, if we have the semantics that *Matric#* is the OID of student and *Matric#* functionally determines *Name* as each *Student* instance only has one name, after getting an answer, we can stop searching the rest of the document. Other situations that semantics improves XML query processing are shown in [24].

XML keyword search. Semantics-based XML keyword search approaches have been proposed to improve search efficiency and quality, such as [23]. However, the use of semantics in current works is still shallow (only on object level, without touching the relationship). For some queries, e.g., {CS5201, CS5208}, only by discovering that there is a relationship type between the object classes student and course, one can infer the meaningful answer of this query should be all the students taking the two courses, e.g., student with *Matric#* being 'HT001' is one of the answers. Otherwise, the root node will be returned, as in most XML keyword search approaches.

Schema/data integration. Most existing approaches (e.g., [1, 9]) integrate elements in different XML schemas based on their structural and linguistic similarities. However, this information is still coarse-grained. In the corresponding XML schema of XML data tree shown in Fig. 1, *Grade* is an attribute of the relationship type between *Course* and *Student*, though it structurally appears at the same position as attributes of *Student*, such as *Name*. If we do not have this semantics, when we integrate this schema with another schema, in which *Student* has an object attribute *Grade* which means the numbering of the year a student has reached in school, we may wrongly integrate these 2 different attributes which have the same attribute name *Grade*, because of their high structural and linguistic similarities.

Although semantics-based XML processing is attracting more and more research attention, unfortunately, most practical applications are still semantics-less. The main issue here is the availability of such semantic information. Most existing XML schema languages used by applications, e.g., DTD and XSD, cannot fully represent the semantics such as object class, relationship type, OID,

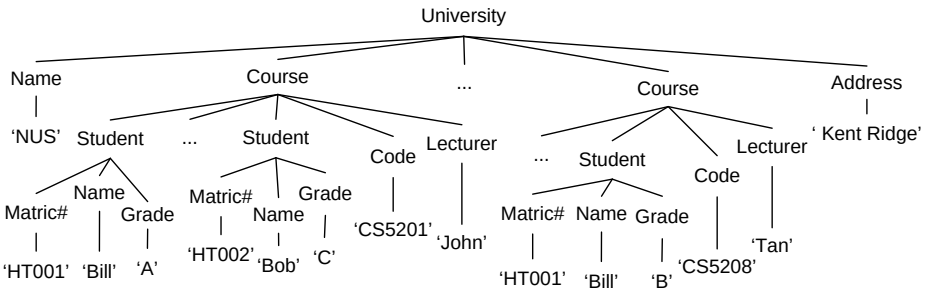


Fig. 1. An XML data tree

object attribute and relationship attribute. Despite the existence of semantically rich XML models, e.g., ORA-SS [10], such model still requires manual provision of semantics from the initial design or during model transformation. We believe that only if the automatic semantics discovery technique is developed to a satisfactory level, the research achievements in, e.g., semantics-based query optimization, semantics-based keyword search, semantics-based schema/data integration and so on, will be widely adopted by different applications.

In this paper, we propose a novel approach to discover semantics from XML schema automatically. We divide our approach into 3 steps: (1) We propose a heuristic-based decision tree, which uses characteristics of different semantic concepts to distinguish object class from others such as composite attribute, aggregational node and explicit relationship type; (2) We propose heuristic rule which captures characteristics of OID to identify candidate OIDs for each object class identified in step 1. We also collect statistic information of OID from real data, and use data mining approach to train a Bayesian Network ranking model to return a unique OID for each object class. Then, object attributes and relationship attributes can be easily distinguished using the identified OID; (3) Having the semantic concepts identified in previous steps, we discovery the implicit relationship types which are not shown in the XML schema tree with the help of functional/multi-valued dependency discovered from the XML data.

The main contributions of this paper include:

1. We reveal the importance of semantics in various XML applications including query processing, keyword search and schema/data integration, and show how semantics can help to increase their efficiencies and accuracies. Our XML semantic discovery approach will work as a foundation for all semantics-based XML models and semantics-based XML query processing, semantics-based XML keyword search and schema/data integration techniques.
2. We propose a novel approach to automatically discover semantics from XML schema. Different from the existing semantics inference approaches mentioned in [2, 11], which only identify objects in XML data or object classes in XML schema, our approach considers a more comprehensive set of se-

mantic concepts. In particular, we also discover OID, relationship type and distinguish between object attribute and relationship attribute.

3. To validate our semantic discovery approach, we conduct experiments over 15 real world data-centric XML schemas and synthetic XML schema from 5 real world relational data sets, to show that the semantics discovered by our approach has high accuracy.

The rest of this paper is organized as follows. In Section 2, we clarify the concept of semantics used in our approach. In Section 3, we present our automatical semantics discovery approach. In Section 4, we conduct experiments to show the quality of our approach with different measurements. Related work is reviewed in Section 5. Conclusion and future work are shown in Section 6.

2 Preliminary

In this section, we describe the concepts used in our approach. First, similar to the XML data model, we also model an XML schema as a tree, which captures the structure information of XML schema. For ease of description, all the following concepts are defined on a schema tree, called *XML schema tree*. Some concepts are adopted from ORA-SS model [10].

In XML schema tree, there are two types of node: *leaf node* and *internal node*. *Object class* is an internal node which represents a real word entity or concept. An object class has a set of *object attributes* as its child nodes or descendant nodes, to describe its properties. Each object class also has an *object identifier (OID)* in its attribute set to uniquely identify each instance. Several object classes may be connected through a *relationship type* which may or may not explicitly appear in the XML schema tree. We call those relationship types which explicit appear in the XML schema tree as *explicit relationship type* and those not appear as any internal node in XML schema tree as *implicit relationship type*. A relationship type may also have a set of *relationship attributes*. Moreover, we also notice the *identifier dependency (IDD) relationship type* between a normal object class and a *weak object class*, which is similar to the *weak entity type* in ER model. If a relationship type explicitly appears as an internal node, its *relationship attributes* will appear as its child nodes or descendant nodes. Otherwise, its relationship attributes normally appear under the lowest object class that participates in that relationship type. In XML schema tree, *aggregational node* and *composite attribute* are also represented as internal node. An aggregational node aggregates child nodes with identical/similar meaning, and serves as a structural node without extra meaning besides the meaning of its child nodes. A composite attribute is a special kind of object/relationship attribute that contains multiple components, each of which can be a single attribute or another composite attribute.

Based on the concepts mentioned above, we define *ORA-Semantics*, which is the scope of the semantics we consider in this paper.

Concept 1 ***ORA-semantics (Object-Relationship-Attribute-semantics)*** *In an XML schema tree, ORA-semantics is the identification of object class, object identifier (OID), object attribute, aggregational node, composite attribute and explicit/implicit relationship type with relationship attributes.*

All semantic concepts contained in ORA-semantics can also be found in ER model. For decades these semantic concepts have been proved to be useful and they have helped database designer to easily understand the semantics and design the corresponding relational model.

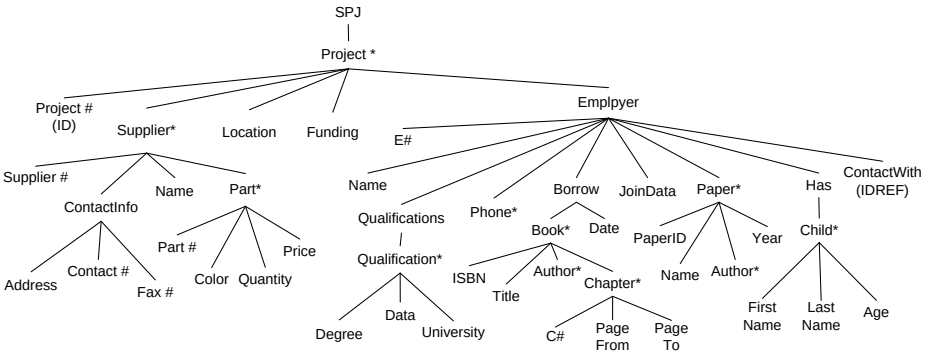


Fig. 2. An XML schema Tree

Example 1. Fig. 2 shows an XML schema tree. We can infer the internal nodes *Project, Supplier, Part, Employee, Book, Paper* and *Child* are object classes, with their OIDs *Project# (specified as ID attribute), Supplier#, Part#, E#, ISBN, PaperID* and *FirstName, LastName* respectively. The internal node *Borrow* is an explicit relationship type between object classes *Employee* and *Book* with relationship attribute *Date*, and *Has* is another explicit relationship type between *Employee* and *Child* without relationship attribute. The leaf node *price* is a relationship attribute of the binary relationship type between the *Supplier* and *Part*, and the leaf node *Quantity* is a relationship attribute of the ternary relationship type among *Project, Supplier* and *Part*. There is an identifier dependency(IDD) relationship type between object classes *Book* and *Chapter*, which makes *Chapter* a weak object class with its OID {*ISBN, C#*}. Also, the internal nodes *ContactInfo* and *Qualification* are composite attributes and the internal node *Qualifications* is an aggregational node. All other leaf nodes are object attributes of the corresponding object classes.

3 Semantics Discovery

We uses heuristic approach to discover the ORA-semantics with XML schema and data as input. All heuristics in our approach is summarized based on the characteristics of different ORA-semantics. Some of these characteristics are directly discovered from the XML schema, some are based on the common way of designing the XML schema and data, and some are discovered using data mining approach in given XML data.

XML schema summarization/extraction has been studied in [6], in case a schema is not available alongside an XML data. For an input XML schema tree, we need its structural information, tag names of its nodes, and ID/IDREF(S) attributes if any; while for the data, we need the functional dependency(FD) and multi-valued dependency(MVD) constraints in it. The general process of our approach, is shown in Fig. 3. Preprocessor takes XML data and XML schema (if available) as input and returns all discovered FDs/MVDs, statistic information and XML schema, which are used as inputs of our main procedure. As mentioned before, our approach can be separated into three steps: (1) Internal node classification, which uses bottom-up approach to classify all internal nodes in XML schema tree into one of the following kinds of ORA-semantics: object class, composite attribute, aggregational node and explicit relationship type; (2) Leaf node classification, which uses top-down approach to identifies OID for each object class, and then distinguishes object attributes from relationship attributes using the identified OIDs; (3) Implicit relationship type identification, which identifies implicit relationship types, which are not represented in the XML schema. In the following sections, we will describe each step separately.

3.1 Step 1: Internal Node Classification

We classify internal nodes in an XML schema tree into four categories: object class, aggregational node, composite attribute (of either object class or relationship type), and explicit relationship type. Based on the structural and linguistic characteristics of these different kinds of ORA-semantics, we propose a heuristic-based decision tree which can be used to distinguish the previous 4 kinds of ORA-semantics apart. Given an internal node of XML schema tree as input, the leaf node it falls into is the category it is identified as. The heuristic-based decision tree is shown in Fig. 4, and the pseudocode of our internal node classification is given in Algorithm 1. We use bottom-up approach in our algorithm so that the category of an internal node can help to identify the category of its parent internal node. Furthermore, the heuristic-based decision tree shows that our approach for internal node classification is complete, which means all internal nodes in a XML schema tree can be classified into one of the above four categories. For ease of description, we represent the heuristic-based decision tree using following 5 heuristic rules, and the number in the leaf node of the heuristic-based decision tree is the corresponding heuristic rule number that can identifies this kind of ORA-semantic.

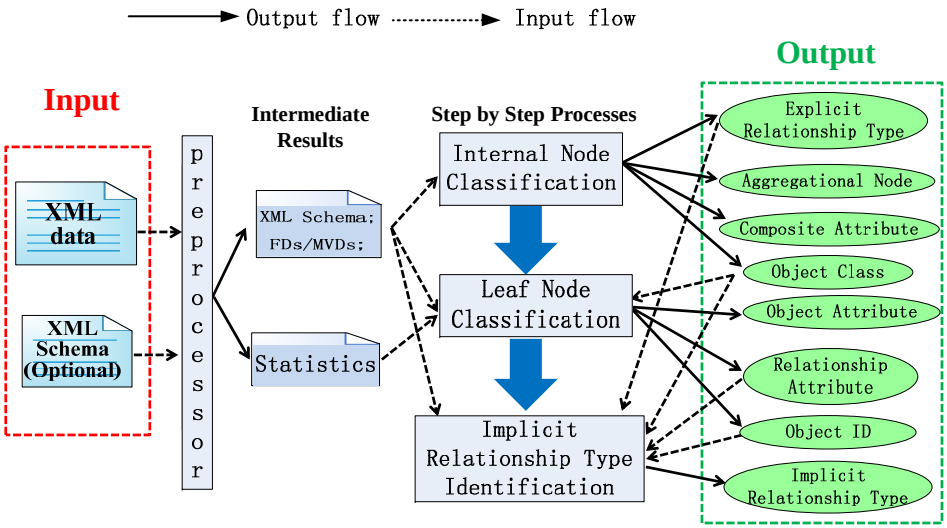


Fig. 3. General process of our automatical semantics discovery approach

Algorithm 1: Internal Node Classification

Input: Set of internal nodes $\mathbb{N}$ in XML schema tree, XML data
Output: Set of identified object classes $\mathbb{N}$, Set of identified composite attributes $\mathbb{C}$, Set of identified aggregational nodes $\mathbb{A}$ and Set of identified explicit relationship type $\mathbb{ER}$

1 **foreach** *internal node* $n \in \mathbb{N}$ *in bottom-up order* **do**
2 Input n into the Heuristic-based Decision Tree;
3 The Heuristic-based Decision Tree returns the classification of n as object class, composite attribute, aggregational node or explicit relationship type.

Rule 1 *[**Object Class** and **OID**] In an XML schema tree, if an internal node has an ID attribute*[1] *specified in its XML schema as its child, then this internal node is an object class, and the ID attribute is the OID of the object class.*

ID attribute specified in the XML schema can be used to identify object class. For example, in the XML schema tree in Fig. 2, the internal node *Project* with the ID attribute *Project#* should be identified as an object class.

Note that some OIDs may not or cannot be specified as ID attribute in the schema because of the limitations of XML DTD, or they are composite attributes. In XML data, the value of ID attribute is required to be globally unique in the document, which makes it impossible for some object classes to have ID attribute being specified in their XML schemas. Although XSD uses key element together with its subelements: selector and field element to specify

[1] ID attribute is specified in DTD. In XSD there is a similar concept key element, which can also be used to identify object class and its OID. For simpleness, Rule 1 is illustrate using ID attribute, but key element also applies.

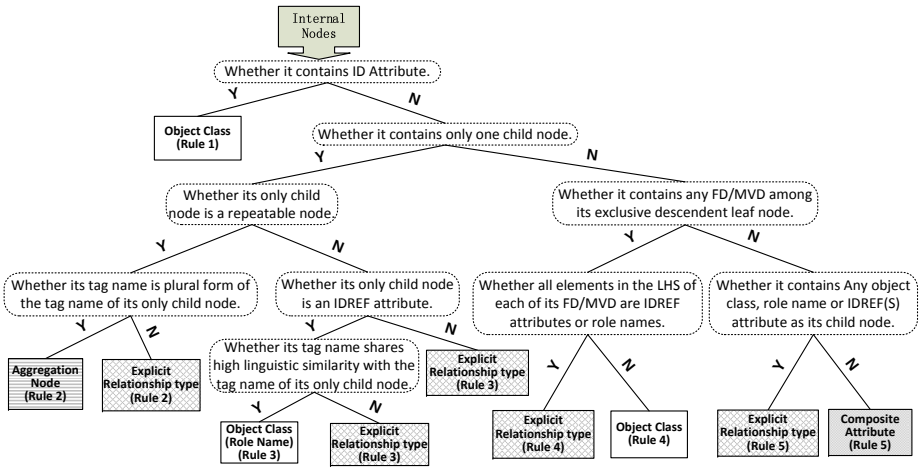


Fig. 4. Heuristic-based Decision Tree for Internal Node Classification

object class and its OID, the same problem still occurs. For example, in Fig. 2, although *Project#* can be specified as OID for object class *Project* using ID attribute in DTD or key element in XSD , both *Supplier#* and *Part#* cannot. Otherwise, a supplier can only supply one project and a part can only be supplied by one supplier, because of the limitations of ID attribute.

Next, we use the following heuristic rules to classify the rest of the internal nodes, including those object classes without ID attribute specified in schema.

Rule 2 *[**Aggregational Node** and **Explicit Relationship Type**] In an XML schema tree, given an internal node* i*, which contains only one child node* c *and* c *is a repeatable node*[2]*. If the tag name of* i *is plural form*[3] *of the tag name of* c*, then* i *is an aggregational node, else it is an explicit relationship type.*

In Rule 2, the first condition is to exclude composite attribute and object class while the second one is to distinguish aggregational node from relationship type. Recall that aggregational node is a structural node for aggregating its repeatable child node. Thus, based on the characteristic of aggregational node, it is reasonable to have its tag name being plural form of the tag name of its child node. For example, in Fig. 2, the internal node *Qualifications* should be identified as an aggregational node, which aggregates its child node *Qualification*. On the other hand, an explicit relationship type may also contain only one child node and the child node is repeatable, such as node *Has* Fig. 2, but the tag name of a relationship type should not be plural form of the tag name of its child node.

[2] Repeatable node is the node which can occur multiple times in the corresponding XML data.

[3] Plural form also includes appending *'s'* to an abbreviation, such as *quals* is the plural form of *qual*, which is the abbreviation of *qualification*.

Concept 2 ***Role Name*** *In an XML schema tree, role name is a special kind of object class, which uses IDREF(S) attribute to refer to the original object class which stores all information about that object class.*

Rule 3 *[**Role Name** and **Explicit Relationship Type**] In an XML schema tree, given an internal node* i*, which contains only one child node* c *and* c *is not a repeatable node. If* c *is an IDREF(S) attribute and the tag name of* i *shares high linguistic similarity with the tag name of* c*, then* i *is the role name of some object class, else* i *is an explicit relationship type.*

Being a special kind of object class, role name has the following two characteristics: (1) it should has only one child node; (2) the child node is an IDREF(S) attribute. For example, in Fig. 5(a), two internal nodes *Landlord* and *Tenant* are the role names of object class *Person*, and they both have an IDREF attribute referring to the object class *Person* as their child node. However, just using these characteristics is not enough to correctly identify role name. For example, in Fig. 4(b) the node *RentBy* also has an IDREF attribute as its only child node. However, the internal node *RentBy* is an explicit relationship type involving the object class its child node referring to. We also notice that the difference between role name and explicit relationship type is that the tag name of role name shares high linguistic similarity with the tag name of its child node, while explicit relationship type does not. This is because the role name and the IDREF(S) attribute represent the same object class. Some researches [12, 22, 19] have already been done on comparing linguistic similarity between different words, such as using information from WordNet[4] for linguistic comparison.

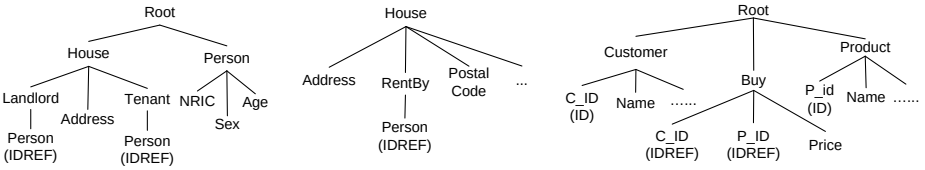


Fig. 5. Different kinds of internal nodes with IDREF(S) in XML schema tree

Concept 3 ***Exclusive Descendant Leaf Node (EDLN)*** *In XML schema tree, an exclusive descendant leaf node of an object class* o *is a leaf node, which is also a descendant node of* o*, but not a descendant node of another object class which is a descendant node of* o*.*

Rule 4 *[**Explicit Relationship Type** and **Object Class**] In XML schema tree, given an internal node* i *with multiple child nodes and there is at least one FDs or MVDs among its exclusive descendant leaf nodes. If all left hand side (LHS) nodes of those FDs or MVDs are IDREF attributes or role names, then* i *is an explicit relationship type, else* i *is an object class.*

[4] http://wordnet.princeton.edu/

The intuitive meaning of EDLN is: given an object class o, its EDLN is a leaf node under o, but no other object class between the EDLN and o. Our Rule 4 uses the heuristic that object class should have FD(s)/MVD(s) among its EDLNs, while composite attribute and aggregational node should not have. However, explicit relationship types may have FD(s)/MVD(s) among its EDLNs, such as *Buy* in Fig. 5(c), which contains a FD $\{C\_ID, P\_ID\} \rightarrow \{Price\}$. In order to distinguish between object class and explicit relationship type, Rule 4 tests whether all nodes appearing on the LHS of each FD/MVD are IDREF attributes or role names. This is because if an explicit relationship type has a FD/MVD among its EDLNs, this FD/MVD must involve some participating object class(es), and it need to use IDREF attribute(s) or role name(s) on the LHS of the FD/MVD to indicate them, such as the internal node *Buy* shown in Fig 5(c). Although object class can also have IDREF attribute(s) or role name(s) on the LHS of its FD/MVD, it also needs to contain its own OID, which is not an IDREF attribute or role name.

Rule 4 uses the FDs/MVDs extracted from XML data. FDs/MVDs in XML data, defined in [8], is not exactly the same as FD/MVD in relational data. In XML data, FD/MVD is valid only under a header path, which is a path expression starting at the root and ending at a common ancestor node of all nodes involved in the FD/MVD. For example, in Fig. 2, given a header path */SPJ/Project/Supplier*, we have the FD $\{Supplier\#\} \rightarrow \{Name\}$, and the *Name* under *Employee* should not be considered. In our paper, we do not focus on discovering FD/MVD in XML data, as it has been studied in [16, 26, 27]. For simplicity, in the rest of this paper, given a FD/MVD in XML schema tree, we use the path ending at the lowest common ancestor node of all nodes involved in the FD/MVD as its header path, and do not show it explicitly.

Rule 5 *[**Explicit Relationship Type** and **Composite Attribute**] In an XML schema tree, given an internal node* i *with multiple child nodes and there is no FD/MVD among its exclusive descendant leaf nodes. If* i *has object class, role name or IDREF(S) attribute as its child node, then* i *is an explicit relationship type, else* i *is a composite attribute.*

A composite attribute should has more than one child node and does not have any FD/MVD among its EDLNs, such as *ContactInfo* and *Qualification* in Fig. 2. However, above two conditions cannot exclude some explicit relationship types, such as *Borrow* in Fig. 2, which also has more than one child node *Book* and *Data* and no any FD/MVD among its exclusive descendant leaf node *Date*. Therefore, Rule 5 uses one more condition to distinguish between composite attribute and explicit relationship type, which is explicit relationship type should have at least one object class, role name or IDREF(S) attribute as its child node, while composite attribute should not. This is because beside the ancestor object class, explicit relationship type should also have at least one involving object class as its child node, while composite attribute should not have such characteristic.

3.2 Step 2: Leaf Node Classification

A. OID Discovery After processing all the internal nodes in step 1, our next step is to identify OID of each identified object class. As stated in Rule 1, OID can be explicitly specified in XML schema using ID attribute. Thus in this section, we only consider the case that the single attribute OID is not specified in XML schema (e.g., *ISBN* of object class *Book* in Fig. 2), or the OID contains multiple attributes (e.g., {*FirstName*, *LastName* of object class *Child* in Fig. 2). Before we explain how we identify these OIDs, let us introduce a concept named *Super OID* first.

In an XML schema tree, the attributes under an object class may be its object attributes or attributes of the relationship type which it participates in. Based on the definition of OID, only its object attributes can be functionally/multi-valued determined by its OID, while relationship attributes cannot. Our approach identify OID for each identified object class based on this heuristic. To do this, we introduce a concept named *Super OID* first.

Concept 4 ***Super OID*** *The super OID of an object class o is a minimal set which contains subset of exclusive descendant leaf nodes of o and OIDs of some ancestor object classes of o. Super OID of o should functionally/multi-valued determines all exclusive descendant leaf nodes of o.*

In an XML schema tree, given an object class o, its exclusive descendant leaf nodes (EDLNs) may be object attributes of o, or relationship attributes of some relationship type which o participates in. Based of the definitions of super OID and OID, the former one should functionally/multi-valued determine both object attributes and relationship attribute (if any), while the latter one can only functionally/multi-valued determine the object attributes. The rationale of our approach to identify OID is that given an object class o, its OID and OIDs of some of (zero or more) its ancestor object class form an attribute set S, which also functionally/multi-valued determine all EDLNs of o (including both object attributes and relationship attributes). In case of no relationship attribute being EDLN of o, no OID of ancestor object class of o will be included in S. Recall the definition of super OID, the attribute set S should also be a super OID of o, which is also a superset of OID of o. Thus, this heuristic shows us a way to identify the OID of an object class by excluding all OIDs of its ancestor object classes from its super OID, and what is left should be its OID.

Based on this heuristic, we proposed a top-down approach (as OID of an ancestor object class may be needed for identifying OIDs of its descendant object classes) shown in Algorithm 2 to identify OID for each identified object class without ID attribute being specified in its XML schema. Given an object class o, we create a set $SupED_o$, which is a superset of its exclusive descendant leaf nodes $ExcDesc(o)$, also including OIDs of its ancestor object classes. In $SupED_o$, we identify the super OID, which is a minimal subset of $SupED_o$ that functionally/multi-valued determines all nodes in $ExcDesc(o)$. Also, we use the heuristic that if the OID of an ancestor object class o_j is in an super OID of o,

then the OIDs of all object classes between o_j and o in their XML schema tree should also be included in that super OID. This is because the super OID of o involves the OID of object class o_j means there may be at least one relationship attribute in the exclusive descendant leaf node of o, and there is a corresponding relationship type between o_j and o, and all object classes between them must also participate in that relationship type. As there may be more than one minimal subset of $SupED_o$ that satisfy this condition, there may be more than one super OID for o. For each super OID, we can get an OID candidate for o, by excluding all OIDs of ancestor object classes of o from the super OID. For the object class without ancestor object class, as it should not have any relationship attribute in its exclusive descendant leaf nodes, its super OID will be the same as its OID. In Algorithm 2, we use $AD(o, o')$ to represent that o is an ancestor node of o' in their XML schema tree.

Algorithm 2: Candidate OID Discovery

Input: Identified object classes $\mathbb{O}$, EDLNs, $ExcDesc()$ for each identified object class
Output: OID id_o for each identified object class $o \in \mathbb{O}$

foreach *identified object class* $o \in \mathbb{O}$ **do**
  $SupED_o$=$ExcDesc(o)$; //$SupED_o$ is a super set of exclusive descendant nodes of o
  foreach $o_i \in \mathbb{O}$, *which is ancestor object class of* o **do**
    $SupED_o = SupED_o \cup id_{oi}$; //$id_{oi}$ is the OID of object class o_i
  foreach $SID_o \subset SupED_o$ **do**
    if $\forall e \in ExcDesc(o)$, *such that* $EID_o \rightarrow e$ *or* $SID_o \twoheadrightarrow e$ **then**
      if $\nexists S \subset SID_o$, *such that* $\forall e \in ExcDesc(o)$, *such that* $S \rightarrow e$ *or* $S \twoheadrightarrow e$ **then**
        if $\forall o_j \in \mathbb{O}$ *with its OID* $id_{oj} \in EID_o$, *such that* $\exists o_k \in \mathbb{O}$ *with its OID* id_{ok}, $AD(o_j, o_k)$ *and* $AD(o_k, o)$, *then* $ID_{ok} \in SID_o$ **then**
          foreach $e \in SID_o$ **do**
            if $\nexists o_p \in \mathbb{O}$ *such that* e *is the OID of* o_p **then**
              $e \in id_o$;
          return id_o as an OID candidate of o.

Example 2. In Fig. 2, considering three identified object classes *Project*, *Supplier* and *Part* with their EDLNs, suppose we only get the following full FDs from the XML data: {*Project#*} → {*Location, Funding*}, {*Supplier#*} → {*Name*}, {*Part#*} → {*Color*}, {*Supplier#, Part#*} → {*Price*} and {*Project#, Supplier#, Part#*} → {*Quantity*}. For object class *Project*, we can identify *Project#* as its OID by Rule 1, because it is an ID attribute. For object class *Supplier*, as the single attribute *Supplier#* functionally determines all its EDLNs, we identify *Supplier#* as its OID, the same as its super OID. For object class *Part*, we combine its EDLNs and OIDs of its ancestor object classes *Supplier* and *Project*, and discover the minimal subsets that functionally determine all its EDLNs to be its super OID, which are {Project#, Supplier#, Part#}, {Supplier#, Part#, Quantity} and {Part#, Quantity, Price}. Then we get {Part#}, {Part#, Quantity} and {Part#, Quantity, Price} as OID candidates of object class *Part*.

As is shown in Example 2, the super OID may not be unique for some object classes, and Algorithm 2 may return more than one OID candidate. Different

OIDs give us different ORA-semantics. For example, for the object class *Part*, choosing {Part#, Quantity, Price} as its OID means it does not have any relationship attribute, while choosing *Part#* means it has two relationship attributes *Quantity* and *Price*. Thus, we propose Heuristics 1 to choose the best OID from all OID candidates returned by Algorithm 2.

Observation 1 ***[OID]*** *In an XML schema tree, given an object class* o*, its OID* id_o *is most likely (yet not a must) to be designed with the following features: (1)* id_o *contains only one child node of* o*; (2) The first child node*[5] *of* o *is (part of)* id_o*; (3)* id_o *contains substring 'Identifier', 'Number', 'Key' or their abbreviations in its tag name; (4)* id_o *has numeral as part of its value in its XML data, and the numerical part is in sequence.*

Heuristic 1 is proposed based on our observation of structural and linguistic characteristics of OIDs designed in real XML schemas. Furthermore, we have following two more observations: (1) the object classes without relationship attribute are more than the object classes with relationship attributes as its EDLNs; (2) for relationship attribute, binary relationship attributes are more than ternary relationship attributes, and so on. Thus, given an object class, the chance of having its super OID, which contains its OID, involving only one object class is higher than the case of involving two or more object classes. Based on these and the heuristics mentioned in Heuristic 1, we collect more than 100 object classes with their OIDs being manually specified in their XML schemas and extract the statistic information mentioned above. Using such statistic information, we train a Bayesian Network ranking model to rank all OID candidates for each object class, and choose the best one as its OID. In Example 2, for the object class *Part*, although both {Project#, Supplier#, Part#}, {Supplier#, Part#, Quantity} and {Part#, Quantity, Price} can functionally determine all its EDLNs, {Project#, Supplier#, Part#} get the highest ranking using our Bayesian Network ranking model. Thus, {*Part#*} is identified as the OID of object class *Part* by our approach. (More details about our Bayesian Network ranking model will be given in the experiment part.)

B. Object Attribute and Relationship Attribute Discovery For an explicit relationship type (discovered in Section 3.1), we identify all its EDLNs as its relationship attributes. For implicit relationship type, its relationship attributes should appear under the lowest object class which participates in it, together with object attributes of that object class. . Thus, we propose Rule 6 to distinguish object attribute and relationship attribute among the EDLNs of each identified object class. We use the characteristic that object attribute should be functionally/multi-valued determined by OID of the object class it belongs to, while relationship attribute should not to differentiate them.

[5] Given an internal node in an XML schema tree, we consider the nodes corresponding to both its first subelement node and first attribute node specified in the XML schema as its first child node.

Rule 6 *[**Object Attribute** and **Relationship Attribute**] Given an object class* o *and its OID, if an exclusive descendant leaf node* e *of* o *can be functionally/ multi-valued determined by the OID of* o*, then* e *is an object attribute of* o*, else it is an attribute of an implicit relationship type which* o *participates in.*

Example 3. In Figure 2, given the object class *Part* with its OID *Part#*, its child node *Color* is functionally dependent on its OID, while *Quantity* and *Price* are not. Thus, we identify *Color* as an object attribute of *Part*, while *Quantity*, *Price* as relationship attributes of some relationship types that *Part* participates in.

3.3 Step 3: Implicit Relationship Type Discovery

After identifying object classes, OIDs, object attributes and relationship attributes in XML schema tree, the last ORA-semantics we will discovery is relationship type. Recall that explicit relationship type can be identified by Rule 2, 3, 4, 5 in Section 3.1. However, there are also implicit relationship types which are not explicitly represented as any node in its XML schema tree. In this section, we classify implicit relationship type into four categories: (1) Implicit relationship type with at least one relationship attribute; (2) Implicit relationship type with IDREF(S) attribute; (3) Implicit relationship type with no relationship attribute and no IDREF(S) attribute, and (4) Identifier Dependency (IDD) Relationship Type [10]. In the following, we will introduce all these four categories one by one.

A. Implicit relationship type with at least one relationship attribute For each relationship attribute discovered in Section 3.2 (except for those being EDLNs of explicit relationship type), there must be an implicit relationship type it belongs to. Based on the heuristic that relationship attribute should be functionally/multi-valued dependent on the OIDs of all involved object classes, Algorithm 3 uses a bottom-up approach to identify the implicit relationship type with its degree, and all involved object classes. We use $r(\mathbb{C})$ to represent the implicit relationship type among all object classes in $\mathbb{C}$, and degree of the implicit relationship type is $|\mathbb{C}|$, which is the number of object classes in $\mathbb{C}$.

For each relationship attribute ra, Algorithm 3 creates a set $SemID_{ra}$ containing the OID of its lowest ancestor object class id_{oi}. We use a bottom-up approach so that in each iteration we add the OID of the lowest ancestor object class that has not been considered along the path from ra to the root, into $SemID_{ra}$. Whenever $SemID_{ra}$ functionally/multi-valued determines ra, we identify the implicit relationship type $r(\mathbb{C})$, to which ra belongs, and return the involved object classes as those object classes whose OIDs are in $SemID_{ra}$ and the degree of the implicit relationship type as the number of involved object classes.

Example 4. In Fig. 2, given a relationship attribute *Price*, object classes *Project*, *Supplier* and *Part*, as well as their OIDs. Using Algorithm 3, we find out that {*Supplier#, Part#*} functionally determines *Price*. Then we know there is an

implicit binary relationship type between *Supplier* and *Part*, with relationship attribute *Price*. For another relationship attribute *Quantity*, {*Supplier#, Part#*} cannot functionally/multi-valued determines it. Then we add in the OID of *Project*, and find out {*Project#, Supplier#, Part#*} functionally determines *Quantity*. Then we know there is an implicit ternary relationship type among *Project*, *Supplier* and *Part*, with relationship attribute *Quantity*.

Algorithm 3: Implicit Relationship Type with Relationship Attribute

Input: Identified relationship attribute $\mathbb{A}$; Identified object classes $\mathbb{O}$, with their OIDs; XML schema tree; XML data
Output: Relationship type $r(\mathbb{C})$ with its involved object classes $\mathbb{C}$ and degree $|\mathbb{C}|$, for each identified relationship attribute in $\mathbb{A}$

1 **foreach** *identified relationship attribute* $ra \in \mathbb{A}$ **do**
2 o_i = the lowest ancestor object class of ra.
3 $\mathbb{C} = \{o_i\}$;
4 $SemID_{ra} = id_{oi}$; //id_{oi} is the OID of object class o_i;
5 **foreach** *identified object class* $o_j \in \mathbb{O}$, *along the path from* o_i *to the root in its XML schema tree in bottom up order* **do**
6 $SemID_{ra} = SemID_{ra} \cup id_{oj}$; //$id_{oj}$ is the OID of object class o_j;
7 $\mathbb{C} = \mathbb{C} \cup \{o_j\}$;
8 **if** $SemID_{ra} \rightarrow ra$ or $SemID_{ra} \twoheadrightarrow ra$; **then break**;
9 return implicit relationship type $r(\mathbb{C})$ to which ra belongs, object classes in $\mathbb{C}$ as its involved object classes and $|\mathbb{C}|$ as its degree;

B. Implicit relationship type with IDREF(S) attribute There is a kind of implicit relationship type, and the designer left a hint by specifying an IDREF(S) attribute under an object class, which refers to other object class(es). In XML schema, if an object class has a child node specified as IDREF(S) attribute, we identify that there is an implicit relationship type between the object class and the object class(es) the IDREF(S) attribute referring to. For some XML schema language (e.g., DTD), we do not know which object class(es) the IDREF(S) attribute referring to. There are two methods to identify the object classes involved in this implicit relationship type: (1) The tag name of the IDREF(S) attribute may share high linguistic similarity with the tag name of the object class(es) it refers to or the corresponding OID(s). We can identify them by research work [12, 22, 19] comparing linguistic similarity. The IDREF(S) attribute works as the role name of the object class(es) it refers to. E.g., given two object classes *Department* and *Staff* with OIDs *Dept#* and *Staff#* respectively, if there is an IDREF(S) attribute under *Staff* with its tag name as *Dept#*, we identify an implicit relationship type between *Department* and *Staff*; (2) If we cannot find high linguistic similarity between the IDREF(S) attribute and any object class or OID, then we can also use the XML data to identify which object class(es) the IDREF(S) attribute referring to. The characteristic of IDREF(S) attribute is that the value range of the IDREF(S) attribute in its XML data must be a subset of the value range of the OID of the object class(es) it refers to. E.g., in Fig. 2, if we know that every value of the IDREFS attribute *ContactWith* is also found as a value of OID of object class *Supplier*, there is a high possibility that there is an implicit relationship type between *Employee* and *Supplier*. This method can also be used to verify the first method.

C. Implicit relationship type with no relationship attribute or IDREF(S) The implicit relationship type with no relationship attribute or IDREF(S) will be discovered by Algorithm 4. We start from the object class o located at the lowest level in its XML schema tree, and discover the implicit relationship type involving o and all other involved object classes, which are ancestors of o.

Algorithm 4: Implicit Relationship Type without Relationship Attribute

```
   Input: Identified object classes 𝕆, with their OIDs; XML schema tree; XML data
   Output: Relationship type r(ℂ) with its involved object classes ℂ and degree |ℂ|
1  foreach identified object class o ∈ 𝕆 do
2      if ∃r which is an implicit relationship type involving o then
3          continue;
4      else
5          ℂ = {o};
6          S = Null;
7          foreach identified object class o_j ∈ 𝕆, along the path from o to the root node in
           its XML schema tree in bottom up order do
8              S = S ∪ id_oj;                    //id_oj is the OID of object class o_j;
9              ℂ = ℂ ∪ {o_j};
10             if S ↠ id_o; then break;          //id_o is the OID of object class o;
11         return implicit relationship type r(ℂ) with object classes in ℂ as its involved object
           classes and |ℂ| as its degree;
```

Given an object class o with its OID id_o, without any implicit relationship type being identified, similar to Algorithm 3, we recursively add the OID of the lowest ancestor object class of o that has not been considered into a set S and check whether S multi-valued determines id_o. Whenever S multi-valued determines id_o, we identify an implicit relationship type $r(\mathbb{C})$, and return its involved object classes and degree. Object classes in $\mathbb{C}$ are those object classes whose OIDs are in S. The rationale of Algorithm 4 is that without relationship attribute, given an object class o', if each of its object instance always corresponds to the same set of object instances of its child object class o, there is an implicit binary relationship type between o' and o. Otherwise, this implicit relationship type must also involve some other ancestor object classes of o. And this corresponding relationship can be captured using multi-valued dependency among their OIDs.

Example 5. In Fig. 2, supposing *Project*, *Employee* and *Paper* are identified as object classes, if the same object instance of *Employee* under different *Project* instance always has the some set of *Paper* instances under it in its XML data, then we know there is an implicit binary relationship between object classes *Employee* and *Paper*, regardless of its parent object class *Project*. If under different *Project* instances, the same *Employee* instance has different set of *Paper* instances, as there is no more ancestor object class of *Project*, then we know there is an implicit ternary relationship type among object classes *Project*, *Employee* and *Paper*.

D. Identifier Dependency (IDD) Relationship Type There is a special kind of relationship type called identifier dependency (IDD) relationship type, which means there is a weak object class that can only be identified together with its lowest ancestor object class. For example, in Fig. 2, the relationship type between object classes *Book* and *Chapter* should be an IDD relationship type, which makes *Chapter* a weak object class. For a weak object class, our heuristic rules in Section 3.1 may temporarily classify it as composite attribute. In order to discovery weak object class and IDD relationship type, we propose the following heuristic rule:

Rule 7 ***[IDD Relationship Type]*** *In an XML schema tree, given an internal node* i *and OID* id_o *of its lowest ancestor object class* o*. If* $\exists$ e*,* e *is an exclusive descendant leaf node of* i *such that (1)* e *cannot functionally/multi-valued determine any other exclusive descendant leaf node of* i*; (2)* e *together with* id_o *can functionally/ multi-valued determine all exclusive descendant leaf nodes of* i*, then* i *is a weak object class and there is an IDD relationship type between* o *and* i*.*

For example, in Fig. 2, *C#* of *Chapter* does not functionally/multi-valued determine any other exclusive descendant leaf nodes of *Chapter*, but *C#* together with OID of its parent object class, {*C#, ISBN*} can functionally determine all of them. Thus *Chapter* is a weak object class, having and IDD relationship with its parent object class *Book*.

4 Experiment

In this section, we test the quality of our approach for discovering ORA-semantics in given XML schema trees in term of precision, recall and F-measure[6]. The experimental data includes 15 real world data-centric XML schemas, e.g., the Auction data, the Modial data, the NBA basketball player data, etc. Due to the fact that most practical databases are still in relational model and most XML data are actually translated from relational data and published/exchanged in XML format, we also collected 5 real world relational data sets e.g., the TPC-H data set, the IMDB data set, etc., and asked 8 PhD students working in the area of XML database to design reasonable XML schemas based on all 5 relational schemas. Finally, our experiments were conducted over all these XML schemas.

Among the XML schemas we collected for experiment, there are some complex ones such as the XMark which has a maximal depth of 6 for its XML schema tree and 25 internal nodes in its schema. For all XML schema we used in our experiment, the average maximal depth is 5 and average number of internal node is 10.7. We ask 8 PhD students working in the area of XML database to manually identify the ORA-semantics in all XML schemas and use them as the

[6] F-measure = 2 * precision * recall/(precision + recall)

ground truth of our experiment. In order to handle the case that different students may have different understandings of what a node in XML schema should be identified as, we adopt the probability theory and use the probability of a node being identified as each kind of ORA-semantic as its ground truth rather than using only one ground truth for each node being identified. For example, the ground truth of an internal may be it has 75% to be an object class and 25% to be a composite attribute because among 8 PhD students, 6 of them identify it as an object class and 2 of them identify it as a composite attribute. Thus, both object class and composite attribute will be consider as the ground truth with different probability by considering their expected values in our calculation of precision and recall.

4.1 Accuracy of Internal Node Classification

In order to discovery the ORA-semantics for internal nodes in XML schema trees, besides using the heuristic rules proposed in Section 3.1, we also proposed another approach to classify the internal nodes. We use the structure and linguistic characteristics used in our heuristic rules together with data mining techniques to train a classification model to classify internal nodes into object class, explicit relationship type, aggregational node and composite attribute. We will compare the accuracy of our heuristic rules with the trained classification model. We choose the frequently used decision tree algorithm C4.5 [14] to build our classification model.

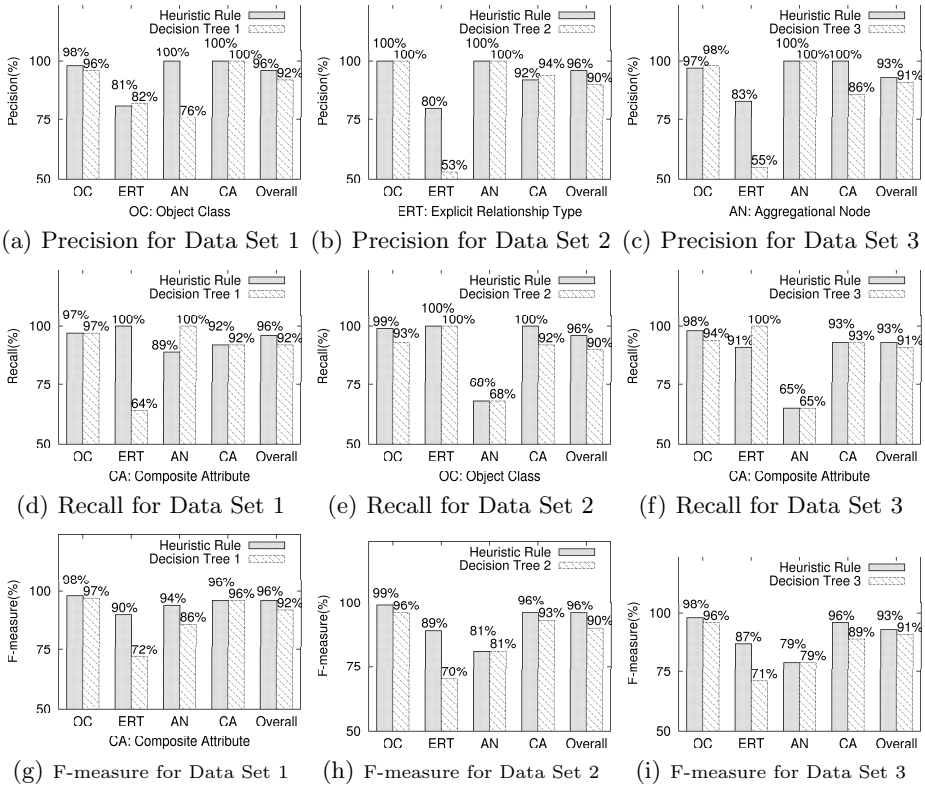
There are totally 505 internal nodes in our input XML schema tree, with their ORA-semantics been labeled (i.e., object class, explicit relationship type, aggregational node or composite attribute). In order to avoid bias, we equally divided all internal nodes into 3 groups, and use 2 of them for training and the rest internal nodes for testing, which will return us 3 classification models (may or may not be the same). We conduct our experiment over our heuristic rules and these classification models using the same testing data set and compare their accuracies in term of their precision, recall and F-measure[7].

As can be seen in Fig. 6, the overall accuracy of both our heuristic rules and the classification models can achieve above 90% of precision, recall and F-measure. And our heuristic rules work even better than the classification models, especially for discovering the explicit relationship types. This is because the explicit relationship type can be designed more flexible with different structure in XML schema tree. Also, when we look into the decision trees of classification models, we found the decision trees trained from different training data set are quite different and most of their branches are not as meaningful as our heuristic rules, which shows that they are heavily dependent on the training data.

4.2 Accuracy of Leaf Node Classification

We also evaluate the accuracy of our approach for discovering OIDs, object attributes and relationship attributes. Recall that our algorithm 2 will return

[7] F-measure = 2 * precision * recall/(precision + recall)



(a) Precision for Data Set 1 (b) Precision for Data Set 2 (c) Precision for Data Set 3

(d) Recall for Data Set 1 (e) Recall for Data Set 2 (f) Recall for Data Set 3

(g) F-measure for Data Set 1 (h) F-measure for Data Set 2 (i) F-measure for Data Set 3

Fig. 6. Precision, recall and F-measure of internal node classification

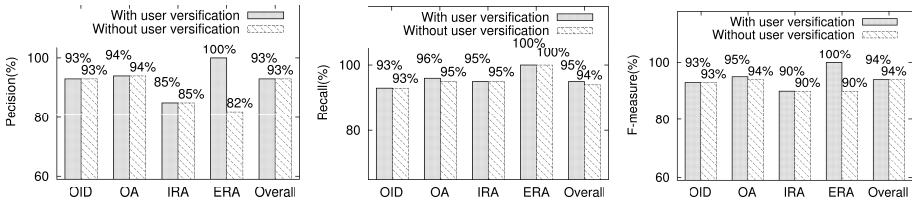
more than one OID candidates for each object class. Then we build a Bayesian Network [5] to rank all those OID candidates, and choose the highest ranked candidate as OID. To be more precise, there are 313 object classes with their OIDs in our data sets. We equally divided them into 3 groups, and randomly choose two of them for training and the rest OIDs for testing. From the labeled OIDs in training data, we calculate the statistics of the features mentioned in Heuristic 1, and build a Bayesian Network, which will return us the probability of an OID candidate to be the correct OID based on the statistics. In Table 1, based on the features mentioned in Heuristic 1 in Section 3.2, we show the top 10 probabilities of bing OID with different features.

Because in our step-by-step approach, some of output of previous step will work as the input for latter step, the accuracy of latter step is depend on the accuracy of previous step. In order to show the accuracy of each step separately, we conduct two groups of experiments to evaluate the precision, recall and F-measure of our approach for leaf node classification, one with user verification, which means all object classes have been correctly labeled in XML schema trees,

Table 1. Features of top 10 ranking of being OID

Rank	Number of Object Attribute	Number of Involved Level	Contains First Child Node or Not	Contains Keyword Substring or Not	Numerical Value with Pattern	Probability of Being OID
1	1	1	T	T	T	0.977
2	1	1	T	T	F	0.977
3	1	1	T	F	T	0.932
4	1	1	T	F	F	0.931
5	1	2	T	T	T	0.876
6	1	2	T	T	F	0.876
7	1	1	F	T	T	0.758
8	1	1	F	T	F	0.758
9	2	1	T	T	T	0.692
10	1	2	T	F	T	0.689

and one based on the results of our internal node classification without user verification, in which some identified object classes may not be correct.



(a) Precision (b) Recall (c) F-measure

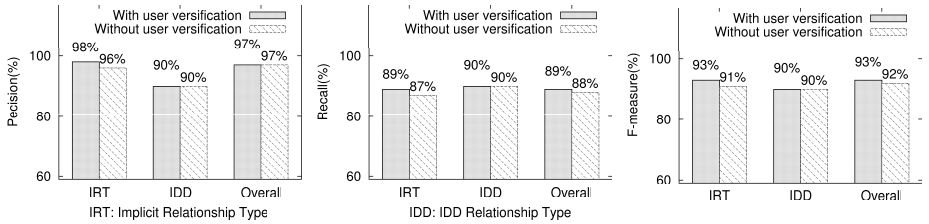
Fig. 7. Precision, recall and F-measure of leaf node classification

As can be seen in Fig. 7, the overall accuracy of our approach for leaf node classification can achieve above 90% of precision, recall and F-measure. Also even without the user verification, the precision and recall of our approach only drops slightly, because our previous step of discovering object class also gets high precision, recall and F-measure.

4.3 Accuracy of Implicit Relationship Type Discovery

Last, we conduct our experiment on our approach for implicit relationship type discovery. Recall that all explicit relationship types have been discovered using our Rule 2, 3, 4, 5 in Section 3.1, and its accuracy has been shown in Fig. 6. Similar to leaf node classification, we also conduct two groups of experiment to evaluate the accuracy of our approach, one with user verification, which means all object classes with their OIDs, object attributes and relationship attributes have been correctly labeled in XML schema trees, and the other based on the results of our previous two steps without user verification.

Fig. 8 shows the precision, recall and F-measure for identifying regular implicit relationship types and IDD relationship type in XML schema trees.



(a) Precision (b) Recall (c) F-measure

Fig. 8. Precision, recall and F-measure of implicit relationship type discovery

Furthermore, we show the overall precision, recall and F-measure of our approach discovering all ORA-semantics in Fig. 9.

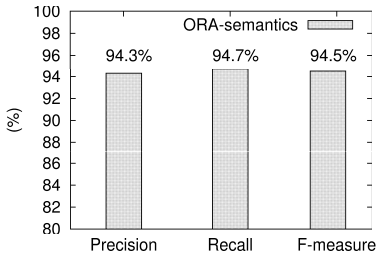


Fig. 9. Precision, recall and F-measure of discovering all ORA-semantics

5 Related Work

To our knowledge few researchers have frontally addressed the problem of automatically discovering the implicit semantics embedded in XML data and Schema. The authors of [7] try to find semantic annotations for XML schemas using an ontology. The intention is to discover the meaning of each XML schema node. In contract, our work focuses on a comprehensive set of semantic concepts grouped under the ORA-semantics. Previous work related to the ORA-semantics has mainly focussed on identifying objects in XML data. [15] introduces an approach to identify object with approximate joins. In [21], XML objects are compared using string comparison functions to detect duplicate objects. The set of semantic concepts targeted in these works is limited. The work above are also stricter to XML data and do not exploit the availability of XML schemas.

In [2], in the context of view design, all internal nodes in an XML schema tree are considered as object class. This is not correct since internal nodes can be relationships, composite attributes or aggregational nodes as well. In the

context of keyword search, XSeek [11] infers semantics from keyword queries to identify return nodes. This work still only infers semantics of objects by using repeatable node. In Web application, The identification of semantic concepts is usually done the manual effort of users, e.g.[18]. For the sake of conciseness, we do not continue this catalogue raisonné of works that address elements of the question but do not provide a global and satisfactory solution. The originality and significance of our work reside in that it addresses and solves the problem holistically and independently from any specific type of target processing.

6 Conclusion and Future Work

The availability of a conceptual schema or of elements of semantics constitute invaluable leverage for improving the effectiveness, and sometimes the efficiency, of many tasks including query processing, keyword search and schema and data integration. Yet XML data and XML schema are instances and schemas of a logical model that fails to explicitly represent the intended semantics.

In this paper we have presented and evaluated a set of techniques for the discovery of semantics implicitly embedded into XML data and schemas. Our target model is of the family of the Object-Relationship-Attribute for Semi-Structured data (ORA-SS) model. We therefore refer to the set of instances of these inferred semantic concepts as the ORA-semantics of the application. The techniques we devised and presented leverage a heuristic-based decision tree in order to identify object classes, explicit relationship types, aggregational nodes and composite attributes from internal nodes of an XML schema tree. The techniques use heuristic rules together with statistic information to identify OID for each object class and to distinguish object attributes and relationship attributes from leaf nodes of an XML schema tree. Finally, our approach is also able to discover the implicit relationship type among object classes. We have empirically evaluated the effectiveness of our proposed approach using several established data sets and schemas. The experiments that we have conducted demonstrate that our approach can achieve almost 95% overall precision, recall and measure.

In our future work, we will use this discovered semantics information to facilitate different XML-related applications. Semantic-based XML query processing will be proposed to improve the efficiency of answer XML queries. We will also propose object-oriented XML keyword search and object-oriented XML schema/data integration. Besides these, we will also try to discovery this semantics information by using not only structure information but also ontology information to further increase our accuracy.

References

1. D. Aumueller, H. H. Do, S. Massmann, and E. Rahm. Schema and ontology matching with COMA++. In *SIGMOD Conference*, pages 906–908, 2005.
2. Y. B. Chen, T. W. Ling, and M.-L. Lee. Designing valid XML views. In *ER*, pages 463–478, 2002.
3. S. Dill, N. Eiron, D. Gibson, D. Gruhl, R. V. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. A. Tomlin, and J. Y. Zien. Semtag and seeker: bootstrapping the semantic web via automated semantic annotation. In *WWW*, pages 178–186, 2003.
4. A. Doan, R. Ramakrishnan, F. C. 0002, P. DeRose, Y. Lee, R. McCann, M. Sayyadian, and W. Shen. Community information management. *IEEE Data Eng. Bull.*, 29(1):64–72, 2006.
5. N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2-3):131–163, 1997.
6. J. Hegewald, F. Naumann, and M. Weis. Xstruct: Efficient schema extraction from multiple and large XML documents. In *ICDE Workshops*, page 81, 2006.
7. J. Kopecký, T. Vitvar, C. Bournez, and J. Farrell. SAWSDL: Semantic annotations for wsdl and XML schema. *IEEE Internet Computing*, 11(6):60–67, 2007.
8. M.-L. Lee, T. W. Ling, and W. L. Low. Designing functional dependencies for XML. In *EDBT*, pages 124–141, 2002.
9. M.-L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: clustering XML schemas for effective integration. In *CIKM*, pages 292–299, 2002.
10. T. W. Ling, M. L. Lee, and G. Dobbie. Semistructured database design. Springer, 2005.
11. Z. Liu and Y. Chen. Identifying meaningful return information for XML keyword search. In *SIGMOD Conference*, pages 329–340, 2007.
12. S. Mizuta and K. Hanya. Specifications of word set in linguistic approach for similarity estimation. In *BICoB*, pages 25–29, 2010.
13. S. Nestorov, S. Abiteboul, and R. Motwani. Extracting schema from semistructured data. In *SIGMOD Conference*, pages 295–306, 1998.
14. J. R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.
15. L. Ribeiro and T. Härder. Entity identification in XML documents. In *Grundlagen von Datenbanken*, pages 130–134, 2006.
16. H. Shi, T. Amagasa, and H. Kitagawa. Fast detection of functional dependencies in XML data. In *XSym*, pages 113–127, 2010.
17. S. B. Siglex, P. Buitelaar, K. Hasida, and C. Universitaire. Semantic annotation and intelligent content, 2000.
18. A. Spink. A user-centered approach to evaluating human interaction with web search engines: an exploratory study. *Inf. Process. Manage.*, 38(3):401–426, 2002.
19. Y. Tang and J. Zheng. Linguistic modelling based on semantic similarity relation among linguistic labels. *Fuzzy Sets and Systems*, 157(12):1662–1673, 2006.
20. V. S. Uren, P. Cimiano, J. Iria, S. Handschuh, M. Vargas-Vera, E. Motta, and F. Ciravegna. Semantic annotation for knowledge management: Requirements and a survey of the state of the art. *J. Web Sem.*, 4(1):14–28, 2006.
21. M. Weis and F. Naumann. Detecting duplicate objects in XML documents. In *IQIS*, pages 10–19, 2004.
22. D. Wu and J. M. Mendel. A vector similarity measure for linguistic approximation: Interval type-2 and type-1 fuzzy sets. *Inf. Sci.*, 178(2):381–402, 2008.

23. H. Wu and Z. Bao. Object-oriented xml keyword search. In *ER*, pages 402–410, 2011.
24. H. Wu, T. W. Ling, and B. Chen. VERT: A semantic approach for content search and content extraction in XML query processing. In *ER*, pages 534–549, 2007.
25. G. Xing and V. Parthepan. Efficient schema extraction from a large collection of XML documents. In *ACM Southeast Regional Conference*, pages 92–96, 2011.
26. C. Yu and H. V. Jagadish. Efficient discovery of XML data redundancies. In *VLDB*, pages 103–114, 2006.
27. C. Yu and H. V. Jagadish. XML schema refinement through redundancy detection and normalization. *VLDB J.*, 17(2):203–223, 2008.
28. X. Zhou, X. Zhang, and X. Hu. Maxmatcher: Biological concept extraction using approximate dictionary lookup. In *PRICAI*, pages 1145–1149, 2006.