Correctness Criteria for Normalization of Semistructured Data

Scott Uk-Jin Lee, Jing Sun, Gillian Dobbie
Department of Computer Science, The University of Auckland, New Zealand
{scott, j.sun, gill}@cs.auckland.ac.nz

Lindsay Groves
School of Mathematics, Statistics and Computer Science
Victoria University of Wellington, New Zealand
lindsay@mcs.vuw.ac.nz

Yuan Fang Li
School of Computing
National University of Singapore
liyf@comp.nus.edu.sg

Abstract

The rapid increase in semistructured data usage has lead to the development of various database systems for semistructured data. Web services and applications that utilize large amounts of semistructured data require data to remain consistent and be stored efficient. Several normalization algorithms for semistructured database systems have been developed to satisfy these needs. However, these algorithms lack the verification that would ensure that data and constraints among the data are not lost or corrupted during normalization. In this paper, we propose a set of correctness criteria for normalization of semistructured data, which require that functional dependencies are preserved, data is not lost, and spurious data is not created during normalization. We use the Z specification language to provide a precise and declarative definition of our criteria.

Keywords*: Formal Specification, Semistructured Data, Normalization, ORA-SS, Z.*

1. Introduction

Semistructured data is a data representation without a rigidly defined structure. Its usage has increased dramatically through some of the fastest growing areas in computer science such as the internet, digital libraries and multimedia data management [18]. This rapid increase has led many web services and applications, which utilize large amounts of semistructured data, to require adequate database systems to store and manage semistructured data. Various database systems have been developed for eXtensible Markup Language (XML) [9, 15], a common representation of semistructured data, to satisfy the demands.

As with widely used database systems such as relational database systems, redundant data stored in XML database systems [1] must be minimized; otherwise, it could cause data inconsistencies and anomalies [12]. Several normalization algorithms have been proposed to minimize redundancies in semistructured database systems by transforming the schema of the semistructured data — for example, the normal form for semistructured data (NF-SS) developed by Wu et al. [19], XML normal form (XNF) developed by Embley and Mok [8], and normal form for XML documents developed by Arenas and Libkin [2]. If such normalization algorithms are not designed correctly, it may be possible for the normalization process to lose or corrupt the meaning of the data as well as the dependencies between the data, The consequences of such inconsistencies would be devastating, especially for databases such as those used in online banking applications, credit card transactions, government's legal applications, and any other applications dealing with critical information.

The existing normalization algorithms for semistructured data lack verification support to ensure preservation of data consistency. For instance, the normalization algorithms described by Wu et al. [19], Embley and Mok [8], and Arenas and Libkin [2], use different methods to minimize redundancies in semistructured database systems. However, without adequate verification, there is no guarantee that these algorithms do not corrupt the meaning of the data. In this paper, we extend our previous work [10] and propose correctness criteria for normalization of semistructured data. The definition of these correctness criteria is based on the semistructured data and its functional dependencies that represents the business logic of the data [7]. The proposed correctness criteria define the concepts and rules for dependency preserving and lossless [7] in the context of semistructured data. The dependency preserving property ensures that the functional dependencies of the data is preserved. The lossless property ensures that the transformed schema does not lose data or create spurious

data. The combination of these two properties provides a complete definition for data equivalence that could be used to verify the correctness of normalization for semistructured data, building on similar concepts used to verify the correctness of normalization in various other database systems.

To define correctness criteria specific to semistructured data, an adequate data modeling language for semistructured data and a formal language must be selected [4, 5, 3]. The former should represent the schema of semistructured data, since the normalization algorithms transform the schema of the data. The latter should allow the correctness criteria to be expressed precisely and effectively. For the data modeling language, we use the Object Relationship Attribute model for Semi-Structured data (ORA-SS) [6, 12] because it is a semantically enriched notation for semistructured data design. For the formal language, we use the Z specification notation [17] since it is very effective in representing declarative definitions. In summary, the major contributions of our approach lie in the following three aspects. Firstly, we propose a set of criteria for verifying the correctness of normalization for semistructured data. Secondly, we use the Z specification language to give a precise and declarative definition of these correctness criteria. Finally, we also provide a basis for the automated verification of semistructured data normalization using the ORA-SS notation.

The rest of the paper is organized as follows. Section 2 briefly describes the ORA-SS notation, normalization for semistructured data, and the Z specification language. Section 3 presents the correctness criteria for normalization of semistructured data, and presents declarative definitions of the dependency preserving and lossless properties using Z notation. Section 4 concludes the paper and describes future work.

2. Background

2.1. ORA-SS Data Modeling Language

The Object-Relationship-Attribute model for Semi-Structured data (ORA-SS) is a semantically enriched data modeling language for database design [6]. It has been used in many XML related database applications, such as storage design, normal form definition, view creation, query execution, and the translation of XML to relational schemas [11]. Briefly, the ORA-SS notation consists of four basic concepts: object classes, relationship type, attributes and references as follows (further details can be found in [12]):

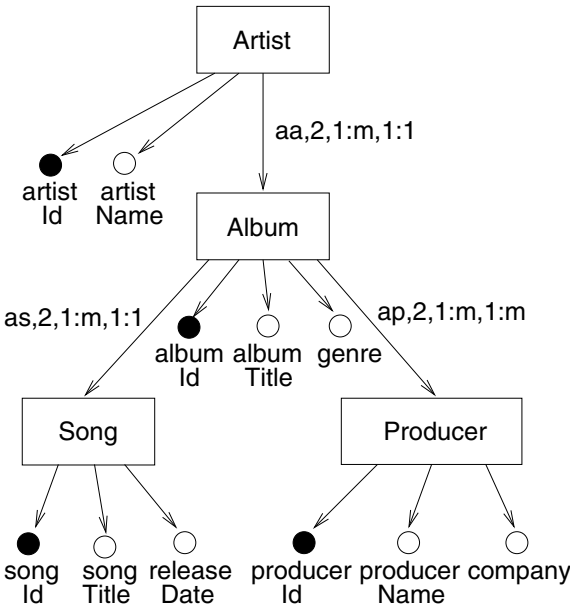
- An *object class* represents an entity type. It is denoted as a labeled rectangle in an ORA-SS diagram.
- A *relationship type* represents a nesting relationship among object classes. It is described by a labeled edge with label (*name*, *n*, *p*, *c*), where the *name* denotes the name of the relationship type, integer *n* indicates the degree of the relationship type, *p* represents the participation constraint of the parent object class in the relationship type, and *c* represents the participation constraint of the child object class in the relationship type.
- An *attribute* represents a property of an object class or relationship type, and is denoted by a labeled circle. An attribute can be a key, indicating that it has a unique value, denoted by a filled circle. Other types of attributes include single valued attributes, multi-valued attributes, required attributes and composite attributes.
- A *reference* indicates that one object class inherits the attributes and relationships of another object class, and is denoted by a dashed line.

For example, Figure 1 presents an ORA-SS schema diagram of a '*Online Music Store*' along with a corresponding XML document. For the simplicity of the example, two assumptions are made to the '*Online Music Store*' schema: (1) a producer can only work for a single company; (2) every song in an album has the same release date. The second assumption excludes the compilation albums such as the 'best-of' albums that contains the songs that are released in different dates.

2.2. Redundancy in Semistructured Data

Semistructured database systems may contain redundancies because of the way that data is represented. For example, in the schema of Figure 1(a), information may be repeated in two places: (1) since every song on an album has the same release date, the value for the '*releaseDate*' attribute of the object class '*Song*' will be repeated for each song on the album; (2) since a producer can produce many albums, the values for all attributes of the object class '*Producer*' may be repeated for many different instances of '*Album*' associated with the same producer. These redundancies can also be observed in the corresponding XML document in Figure 1(b): (1) attribute value '*05-08-1970*' for '*releaseDate*' is repeated for every song on the '*Let It Be*' album; (2) all values for attributes of object class '*Producer*' are repeated in different albums.

These redundancies not only waste storage space, but can also create anomalies when inserting, updating, and deleting data. For instance, one or more instances of '*releaseDate*' for songs that are in the same album could be inserted incorrectly. Similarly, the '*company*' that the same '*Producer*' works for may be inadvertently updated in some places but not others, which corrupts the data by having a '*Producer*' working for many different '*company*'s. This example demonstrates that it is important to have algorithms



(a) An ORA-SS schema diagram for an Online Music Store

```
<Artist artistId = 1>
 <artistName> beatles </artistName>
 <Album albumId = 3>
  <albumTitle> Let It Be </albumTitle>
  <genre> rock </genre>
  ...
  <Song songId = 5>
   <songTitle> Dig It </songTitle>
   <releaseDate> 05−08−1970 </releaseDate>
  </Song>
  <Song songId = 6>
   <songTitle> Let It Be </songTitle>
   <releaseDate> 05−08−1970 </releaseDate>
  </Song>
  <Song songId = 7>
   <songTitle> Maggie Mae </songTitle>
   <releaseDate> 05−08−1970 </releaseDate>
  </Song>
  ...
  <Producer producerId = 1>
   <producerName> George Martin <producerName>
   <company> EMI </company>
  </Producer>
  <Producer producerId = 5>
   <producerName> Phil Spector </producerName>
   <company> Philles Records </company>
  </Producer>
 </Album>
 <Album albumId= 5>
  <albumTitle> Abbey Road </albumTitle>
  <genre> rock </genre>
  ...
  <Producer producerId = 1>
   <producerName> George Martin <producerName>
   <company> EMI </company>
  </Producer>
  ...
```

(b) An XML document

Figure 1. An ORA-SS schema diagram and its XML document

that eliminate or minimize redundancies in semistructured data schemas without corrupting the data.

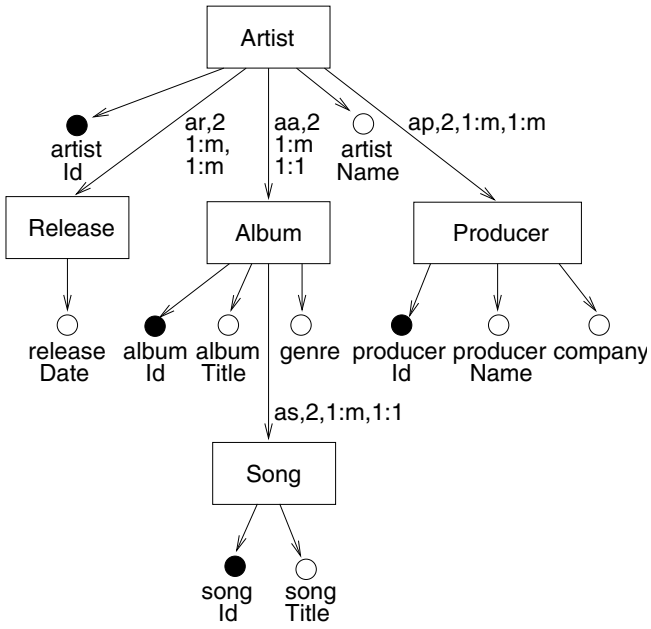
2.3. Normalization

Normalization is the process of analyzing and restructuring a database schema in order to minimize redundancies in data instances. It is considered as one of the most important database design and maintenance processes. To be effective, however, normalization must preserve the semantics of the data. Normalization algorithms for various database systems have been developed and verified to be correct. These algorithms transform the schemas based on the keys and functional dependencies that reflect the constraints on the data [7]. Functional dependencies are usually expressed in the form '$X \rightarrow Y$', which represents the constraint that a set of attributes, 'X', determines another set of attributes, 'Y'.

In the context of relational database schemas, the result of normalization is considered to be semantically equivalent to the original schema if it satisfies two properties: dependency preserving and lossless. The dependency preserving property ensures that each given functional dependency is enforced by the transformed schema; and the lossless property guarantees that no information is lost and no spurious information introduced during the transformation [7].

These two properties can be adapted and extended to the verification of semistructured data normalization. Functional dependencies in semistructured data are written in path notation [2], and have the form '*pathX* → *pathY*', where '*pathX*' represents a set of paths from the root to an attribute or an object class, and '*pathY*' represents a set of paths from the root to an attribute. Paths are used to describe particular object classes and attributes because different object classes or attributes in different places in the schema diagram may have a same name. For example, the following is the given set of functional dependencies for the ORA-SS schema in Figure 1(a).

$\{$'*Artist.@artistId* → *Artist.@artistName*',
'*Artist.Album.@albumId* → *Artist.Album.@albumTitle*',
'*Artist.Album.@albumId* → *Artist.Album.@genre*',
'*Artist.Album.@albumId* → *Artist.Album.@releaseDate*',
'*Artist.Album.Song.@songId*
→ *Artist.Album.Song.@songTitle*',
'*Artist.Album.Producer.@producerId*
→ *Artist.Album.Producer.@producerName*',
'*Artist.Album.Producer.@producerId*



(a) A "normalized" ORA-SS schema diagram for an Online Music Store

```
<Artist artistId = 1>
 <artistName> Beatles </artistName>
 <Release>
  <releaseDate> 11−26−1969 </releaseDate>
 </Release>
 <Release>
  <releaseDate> 05−08−1970 </releaseDate>
 </Release>
 ...
 <Album albumId = 3>
  <albumTitle> Let It Be </albumTitle>
  <genre> rock </genre>
  ...
  <Song songId = 5>
   <songTitle> Dig It </songTitle>
  </Song>
  <Song songId = 6>
   <songTitle> Let It Be </songTitle>
  </Song>
  ...
 </Album>
 <Album albumId= 5>
  <albumTitle> Abbey Road </albumTitle>
  <genre> rock <\genre>
  ...
 </Album>
 ...
 <Producer producerId = 1>
  <producerName> George Martin </producerName>
   <company> EMI </company>
 </Producer>
 <Producer producerId = 5>
   <producerName> Phil Spector </producerName>
   <company> Philles Records </company>
  </Producer>
 </Album>
 ...
```

(b) An XML document

Figure 2. A "normalized" ORA-SS schema diagram and its XML document

$\rightarrow$ *Artist.Album.Producer.@company'*,
'Artist.Album.@albumId$\rightarrow$ *Artist'*,
'Artist.Album.Song.@songId $\rightarrow$ *Artist.Album'*}

These functional dependencies are given separately with the ORA-SS schema to represent the business logics and constraints between '*Online Music Store*' data. For example, the dependency which describes '*albumId*' determines '*releaseDate*' of the ORA-SS schema in Figure 1(a)is represented in path notation as '*Artist.Album.@albumId* $\rightarrow$ *Artist.Album.Song.@releaseDate*' where the '@' symbol is a prefix for attributes.

Using the given functional dependencies, dependency preserving and lossless properties can be applied to normalization algorithms for semistructured data. For example, consider Figure 2(a) which may be the result of normalizing the schema in Figure 1(a). Even though the redundancies are removed as demonstrated in Figure 2(b), we can see that the "normalized" schema in Figure 2(a) is not semantically equivalent to its original form. Firstly, it is not possible for '*albumId*' to determine '*releaseDate*' in the "normalized" schema of '*Online Music Store*'. Consequently, the functional dependency '*Artist.Album.@albumId* $\rightarrow$ *Artist.Album.@releaseDate*' which was given for the original schema is lost. Secondly, the normalized ORA-SS schema in Figure 2(a) introduces spurious data. If the XML data in Figure 2(b) was queried to find out who produced the '*Abbey Road*' album, information about '*George Martin*' and '*Phil Spector*' would be returned. This information is spurious since, according to the original XML in Figure 1(b), '*Abbey Road*' was produced by '*Phil Spector*' alone. This spurious data is introduced by the relationship between '*producer*' and '*album*' being lost. Similarly, the relationship between an '*album*' and its '*releaseDate*' of is lost and produces inconsistent information.

The above example shows that data redundancies in schemas for semistructured data can be removed or minimized by transforming the schema. However, this transformation may not guarantee the preservation of data and its constraints. Therefore, it is essential to have a clear set of criteria for verifying the correctness of normalization for semistructured data.

2.4. Z Formal Specification Language

Z [17] is a formal specification language based on set theory and first-order logic, and has been widely used for providing formal semantics and verification in various ap-

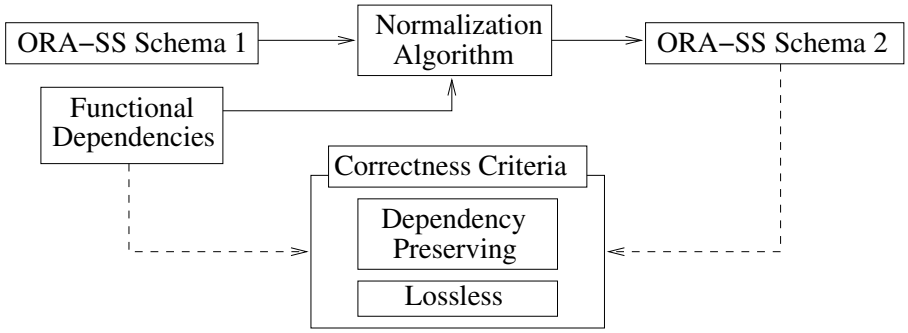


Figure 3. Verification of normalization for semistructured data

plication domains, such as database systems, architecture modeling and so on [14, 16]. We use Z to specify our correctness criteria, because it provides a convenient and well-known mathematical language which can be type checked and analysed with tools such as *fuzz* and Z/EVES [13].

The following types are used to represent ORA-SS schemas. '*Attribute*' represents the set of attributes.

$$[Attribute]$$

'*ObjectClass*' represents object classes, which comprise a primary key, candidate keys and attributes.

$$\begin{array}{|l} \textit{ObjectClass} \\ \hline pkey : Attribute \\ keys : \mathbb{F}_1\, Attribute \\ attributes : \mathbb{F}_1\, Attribute \\ \hline pkey \in keys \\ keys \subseteq attributes \\ \hline \end{array}$$

'*Relationship*' represents relationship types, which comprise object classes and relationship attributes. The '*Relationship*' type used here is simplified by omitting the degree, name and participation constraints, which are not used in defining the correctness criteria.

$$\begin{array}{|l} \textit{Relationship} \\ \hline objectClasses : \mathbb{F}\, ObjectClass \\ attributes : \mathbb{F}\, Attribute \\ \hline \#objectClasses \geq 2 \\ \hline \end{array}$$

'*Schema*' represents an ORA-SS schema, which comprises a set of object classes and corresponding relationships.

$$\begin{array}{|l} \textit{Schema} \\ \hline objectClasses : \mathbb{F}\, ObjectClass \\ relationships : \mathbb{F}\, Relationship \\ \hline \forall\, rel : relationships \bullet \\ \quad rel.objectClasses \subseteq objectClasses \\ \forall\, attr1, attr2 : \\ \quad \bigcup \{objcls : objectClasses \bullet objcls.attributes\} \\ \quad \cup \bigcup \{rel : relationships \bullet rel.attributes\} \bullet \\ \quad\quad attr1 \neq attr2 \\ \hline \end{array}$$

Note that the first assumption presented in Section 3 is enforced in the definition of '*ObjectClass*', since every object class is required to have a primary key ('*pkey*'). The second assumption, that every attribute and object class has a unique name, is enforced as a state invariant in the definition of '*Schema*'.

3. Defining Correctness Criteria for Normalization of Semistructured Data

A normalization algorithm for semistructured data must ensure that the transformed schema is semantically equivalent to the original schema in an appropriate sense. We define a notion of semantic equivalence for semistructured data by adapting the definition used for relational databases discussed in Section 2.3: two schemas are considered semantically equivalent if the expected constraints are modeled in both schemas, and when the schemas are populated with data, no information is lost and no spurious data is created. In our approach, these correctness criteria are applied to the schema represented in the ORA-SS data modeling language and its functional dependencies represented using path notation.

Figure 3 shows the overall approach to verifying that normalization of semistructured data preserves the semantics of the data. A '*Normalization Algorithm*' takes an initial schema ('*ORA-SS Schema 1*') and a given set of '*Func-*

tional Dependencies', and produces a new schema ('*ORA-SS Schema 2*'). The given set of '*Functional Dependencies*' and the transformed '*ORA-SS Schema 2*' are used in the '*Correctness Criteria*' to verify the semantic equivalence of the transformation. The '*Dependency Preserving*' criterion requires that the given set of '*Functional Dependencies*' for '*ORA-SS Schema 1*' will still be preserved in '*ORA-SS Schema 2*', and the '*Lossless*' criterion requires that no data in '*ORA-SS Schema 1*' lost in '*ORA-SS Schema 2*' and no new data is created. In other words, it requires that every set of data that can be obtained from the original '*ORA-SS Schema 1*', and no other, can also be obtained from the transformed '*ORA-SS Schema 2*'.

3.1. Functional Dependencies

In order to verify the correctness of normalization for semistructured data, we need to obtain the set of all functional dependencies implied by the given set of functional dependencies. The standard way to do this is to use Armstrong's Axioms [7], however, Armstrong's Axioms are defined over functional dependencies defined as relations between sets of attributes, i.e. '$X \rightarrow Y$' form. Thus, we must either define and verify a version of Armstrong's Axioms for functional dependencies in path notation, or convert functional dependencies from path notation to '$X \rightarrow Y$' form. We adopt the latter approach for simplicity.

This conversion can be easily achieved without losing or corrupting constraints among data if the following two assumptions are placed on the ORA-SS schema: (1) every object class in the ORA-SS schema has a primary key; (2) every attribute and object class has a unique name. The first assumption eliminates the use of object class in the functional dependencies since primary key can be used to uniquely represent the object class. The second assumption eliminates the need for path notations, since every object class and attribute can be distinguished by its name. It is straightforward to convert any ORA-SS schema into one that satisfies these assumptions; functional dependencies using path notation can then be converted into '$X \rightarrow Y$' form.

For example, the functional dependency '*Artist.Album.Song.@songId* $\rightarrow$ *Artist.Album*' in Figure 1(a) can be converted into '*songId* $\rightarrow$ *albumId*': '*Artist.Album.Song.@songId*' is converted into '*songId*' since the name of the attribute '*songId*' is unique in the schema, and '*Artist.Album*' is converted into '*albumId*' since the attribute '*albumId*' is a primary key that determines the object class '*Album*'.

We define a function to compute the closure of a given set of functional dependencies, in Z, as follows. Firstly, we define type '*FunctionlDependencies*', which represents sets of functional dependencies as binary relations over sets of attributes.

$$FunctionalDependencies == \mathbb{F}_1\, Attribute \leftrightarrow \mathbb{F}_1\, Attribute$$

Next, we define a function to compute the set of functional dependencies that can be obtained from a given set by applying Armstrong's reflexivity, augmentation and transitivity rules [7].

$$\begin{array}{|l} armstrongsAxiom : FunctionalDependencies \rightarrow \\ \quad FunctionalDependencies \\ \hline \forall\, givenFD : FunctionalDependencies \bullet \\ armstrongsAxiom(givenFD) = \\ \quad givenFD \cup \\ \quad \{X, Y : \mathbb{F}_1\, Attribute \mid Y \subseteq X \wedge \\ \qquad X \subseteq getFDAttributes(givenFD) \bullet X \mapsto Y\} \cup \\ \quad \{X, Y, Z : \mathbb{F}_1\, Attribute \mid \\ \qquad X \mapsto Y \in givenFD \bullet X \cup Z \mapsto Y \cup Z\} \cup \\ \quad \{X, Y, Z : \mathbb{F}_1\, Attribute \mid \\ \qquad \{X \mapsto Y, Y \mapsto Z\} \subseteq givenFD \bullet X \mapsto Z\} \end{array}$$

The auxiliary function '*getFDAttributes*' returns the set of attributes appearing in a given set of functional dependencies.

$$\begin{array}{|l} getFDAttributes : FunctionalDependencies \rightarrow \\ \quad \mathbb{F}\, Attribute \\ \hline \forall\, fds : FunctionalDependencies \bullet \\ getFDAttributes(fds) = \\ \quad \bigcup \{fd : fds \bullet fd.1 \cup fd.2\} \end{array}$$

The function '*findClosure*' computes the closure of a set of functional dependencies by repeatedly applying '*armstrongsAxiom*' until a fixed point is reached.

$$\begin{array}{|l} findClosure : FunctionalDependencies \rightarrow \\ \quad FunctionalDependencies \\ \hline \forall\, givenFD : FunctionalDependencies \bullet \\ findClosure(givenFD) = \\ \quad \textbf{if}\ givenFD = armstrongsAxiom(givenFD) \\ \quad \textbf{then}\ givenFD \\ \quad \textbf{else}\ findClosure(armstrongsAxiom(givenFD)) \end{array}$$

3.2. Defining the Dependency Preserving Property

The dependency preserving property ensures that constraints among the data are preserved when the schema is transformed during normalization. To verify correctness of normalization for semistructured database systems, a dependency preserving property must be constructed to suit the semistructured data context.

A semistructured database system can verify whether the dependency preserving property holds by producing the projection of the functional dependencies of the original schema onto the transformed schema. The projection produces the set of functional dependencies from the original schema that can be represented in the new schema. Using the projection, the transformed schema is verified to be dependency preserving if every functional dependency of the original schema is contained in the closure of the projection of the transformed schema. The projection of the given functional dependencies on the transformed ORA-SS schema contains every functional dependency in the closure where both '*X*' and '*Y*' belong to a single object class or to a single relationship of the transformed ORA-SS schema.

The dependency preserving property for semistructured database systems can be defined in the Z formal language as follows. Firstly, we define type '*Boolean*' to represent truth values as 0 (false) and 1 (true).

$$Boolean == \{0, 1\}$$

The '*getRelAttributes*' function returns the set containing all attributes of a given relationship along with the primary keys of object classes that belong to that relationship.

$$getRelAttributes : Relationship \rightarrow \mathbb{F}\, Attribute$$

$$\forall\, rel : Relationship \bullet$$
$$getRelAttributes(rel) =$$
$$\quad rel.attributes \cup$$
$$\quad \{objcls : rel.objectClasses \bullet objcls.pkey\}$$

The '*findProjectedFD*' function produces the projection of the given set of functional dependencies on the ORA-SS schema. The projected functional dependencies are those in the closure of the given set of functional dependencies where both '*X*' and '*Y*' either belong to a single object class of the ORA-SS schema or are attributes of a single relationship.

$$findProjectedFD : Schema \times$$
$$\quad FunctionalDependencies$$
$$\quad \rightarrow FunctionalDependencies$$

$$\forall\, s : Schema;\ givenFD : FunctionalDependencies \bullet$$
$$findProjectedFD(s, givenFD) =$$
$$\quad \{fd : FunctionalDependencies \mid$$
$$\quad\quad (\exists\, objcls : s.objectClasses \bullet$$
$$\quad\quad\quad fd.1 \cup fd.2 \subseteq objcls.attributes) \lor$$
$$\quad\quad (\exists\, rel : s.relationships \bullet$$
$$\quad\quad\quad fd.1 \cup fd.2 \subseteq getRelAttributes(rel))\}$$

The '*isDependencyPreserving*' function checks whether the transformed ORA-SS schema is dependency preserving by testing whether the given set of functional dependencies is contained in the closure of the projection of the functional dependencies on the transformed ORA-SS schema.

$$isDependencyPreserving : Schema \times$$
$$\quad FunctionalDependencies \rightarrow Boolean$$

$$\forall\, s : Schema;\ givenFD : FunctionalDependencies \bullet$$
$$isDependencyPreserving(s, givenFD) = 1 \Leftrightarrow$$
$$\quad givenFD \subseteq$$
$$\quad\quad findClosure(findProjectedFD(s, givenFD))$$

Example 1 The ORA-SS schema in Figure 2(a), which is incorrectly normalized and does not preserve dependency, can be checked using the functions above. According to the '*isDependencyPreserving*' function the transformed ORA-SS schema and the given set of functional dependencies of the example are required for verification. The given set of functional dependencies of the example in Section 2.3 is converted into the '$X \rightarrow Y$' notation and shown below.

{*'artistId → artistName', 'albumId → albumTitle', 'albumId → genre', 'albumId → releaseDate', 'songId → songTitle', 'producerId → producerName', 'producerId → company', 'albumId → artistId', 'songId → albumId'*}

A projection of the given set of functional dependencies on the transformed ORA-SS schema can be produced. Among the given set of functional dependencies, the functional dependencies that are included in the closure of the projection are found and listed below.

{*'artistId → artistName', 'albumId → albumTitle', 'albumId → genre', 'songId → songTitle', 'producerId → producerName', 'producerId → company', 'albumId → artistId', 'songId → albumId'*}

Comparing the two sets of functional dependencies above shows that the functional dependency '*albumId → releaseDate*' in the given set of functional dependencies is not contained in the closure of the projection. Thus, the "normalized" ORA-SS schema in Figure 2(a) is not dependency preserving, since the functional dependency '*albumId → releaseDate*' is not preserved.

3.3. Defining the Lossless Property

The lossless property ensures that no data is lost and spurious data is not created when an ORA-SS schema is transformed during normalization. This means that a query on a dataset based on the transformed schema will produce the same result as a query on the dataset based on the original schema. In another words, the transformed schema is verified to be lossless if the joined set of attributes in the transformed schema produces the same joined set of attributes

as the original schema. The join of the attributes in the transformed schema makes use of the functional dependencies and relationships among attributes. It can be realized either by the attributes belonging to the same object class or relationship, or by the set of attributes that are related through the structure of the schema and joined according to the functional dependencies. Using this correctness criterion, the verification algorithm of the lossless property for a transformed ORA-SS schema is defined and shown below as pseudo code.

```
Let S be the transformed ORA−SS schema
Let OA be the set of attributes that belong to an object class
Let RA be the set of attributes that contains the keys of object
  classes that belong to a relationship and relationship attributes
Let A be the set of every OA for all object classes in S and
  set of RA for all relationships in S
Let FD be the functional dependencies of S in X –> Y notation

Repeat until there is no change to A
    For each functional dependency X –> Y in the closure of FD
        Let Ax be the sets of attributes in A that contains X
        If some set of attributes in Ax contains Y
            For each set of attributes in Ax
                Add Y to the set

If A is same as the set of all attributes in S
    return True
```

The algorithm represents the lossless checking process when sets of attributes in the transformed ORA-SS schema are joined according to their functional dependencies. That is, for each functional dependency '$X \rightarrow Y$' in the closure of '*FD*', if some attribute sets '*Ax*' in '$OA \cup RA$' contains '*X*' and also at least one set in '*Ax*' contains '*Y*', then introduce '*Y*' into every set of attributes in '*Ax*'. Finally, after all the functional dependencies in '*FD*' have been processed, if there exists at least one set of attributes that contains all the attributes in the schema, the transformation is considered to be lossless. Similarly, the lossless property for semistructured data normalization and its verification algorithm can be formally defined in Z as follows.

'*getAllAttributes*' returns the set of attributes in an ORA-SS schema.

$$
\begin{array}{|l}
getAllAttributes : Schema \rightarrow \mathbb{F}\, Attribute \\
\hline
\forall\, s : Schema \bullet \\
getAllAttributes(s) = \\
\quad \bigcup \{objcls : s.objectClasses \bullet objcls.attributes\} \cup \\
\quad \bigcup \{rel : s.relationships \bullet rel.attributes\}
\end{array}
$$

'*getAllAttributeSets*' returns all sets of attributes derived from object classes and relationships in an ORA-SS schema.

$$
\begin{array}{|l}
getAllAttributeSets : Schema \rightarrow \mathbb{F}(\mathbb{F}\, Attribute) \\
\hline
\forall\, s : Schema \bullet \\
getAllAttributeSets(s) = \\
\quad \{objcls : s.objectClasses \bullet objcls.attributes\} \cup \\
\quad \{rel : s.relationships \bullet getRelAttributes(rel)\}
\end{array}
$$

For each functional dependency represented as '$X \rightarrow Y$' in the domain, the '*calculateLossless*' function finds the sets of attributes that contains '*X*' among the given set of sets of attributes and adds '*Y*' to every set of attributes that contain '*X*' if the set of attributes that contains '*Y*' exists among the found sets of attributes that contains '*X*'. The '*calculateLossless*' function recursively performs the process of adding '*Y*' until the given sets of attributes have no change on the addition of '*Y*'. Then the predicate returns the modified sets of attributes. Basically, the '*calculateLossless*' function performs the join on sets of attributes for object classes and relationship of the transformed ORA-SS schema and returns the joined sets of attributes.

$$
\begin{array}{|l}
calculateLossless : \mathbb{F}(\mathbb{F}\, Attribute) \times \\
\quad FunctionalDependencies \rightarrow \mathbb{F}(\mathbb{F}\, Attribute) \\
\hline
\forall\, oldAttrSet, newAttrSet : \mathbb{F}(\mathbb{F}\, Attribute); \\
\quad givenFD : FunctionalDependencies \bullet \\
calculateLossless(oldAttrSet, givenFD) = \\
\quad\quad newAttrSet \Leftrightarrow \\
\quad (\textbf{if } givenFD = \emptyset \\
\quad \textbf{then } newAttrSet = oldAttrSet \\
\quad \textbf{else } (\exists\, fd : givenFD \bullet \\
\quad\quad (\exists\, someAttrSet : \mathbb{F}\, \mathbb{F}\, Attribute \bullet \\
\quad\quad\quad someAttrSet \subseteq oldAttrSet \wedge \\
\quad\quad\quad (\forall\, attrs1 : someAttrSet \bullet fd.1 \subseteq attrs1) \wedge \\
\quad\quad\quad (\exists\, attrs2 : someAttrSet \bullet fd.2 \subseteq attrs2) \Rightarrow \\
\quad\quad\quad\quad newAttrSet = calculateLossless(\\
\quad\quad\quad\quad\quad (oldAttrSet \setminus someAttrSet \\
\quad\quad\quad\quad\quad \cup \{attrSet : \mathbb{P}\, Attribute \mid \\
\quad\quad\quad\quad\quad\quad \mathbb{F}\, attrs : someAttrSet \bullet \\
\quad\quad\quad\quad\quad\quad\quad attrSet = attrs \cup fd.2\}), \\
\quad\quad\quad\quad\quad (givenFD \setminus \{fd\}))))))
\end{array}
$$

'*isLossless*' determines whether there is a set of attributes in the joined sets of attributes for the transformed ORA-SS schema which is equal to the set of all attributes in the transformed ORA-SS schema.

$$
\begin{array}{|l}
isLossless : Schema \times FunctionalDependencies \\
\quad \rightarrow Boolean \\
\hline
\forall\, s : Schema;\ fd : FunctionalDependencies \bullet \\
isLossless(s, fd) = 1 \Leftrightarrow \\
\quad getAllAttributes(s) \in \\
\quad\quad calculateLossless(getAllAttributeSets(s), \\
\quad\quad\quad findClosure(fd))
\end{array}
$$

Example 2 The ORA-SS schema in Section 2.2 Figure 2(a), which could be the result of an normalization algorithm, can be examined using the function defined above. According to the '*isLossless*' function defined in Z, the set of attributes for each object class or relationship of transformed ORA-SS schema is derived and joined according to the given functional dependencies. The following are the sets of attributes derived for every object class and relationship of the transformed ORA-SS schema.

{{*artistId, artistName*}, {*releaseDate*}, {*albumId, albumTitle, genre*}, {*producerId, producerName, company*}, {*songId, songTitle*}, {*artistId*}, {*artistId, albumId*}, {*artistId, producerId*}, *and* {*albumId, songId*}}

This derived sets of attributes were joined recursively according to the predicates of the '*calculatesLossless*' function. For instance, the functional dependency '*albumId* → *albumTitle*' will add attribute '*albumTitle*' to the 3 different sets of attributes '{*albumId, albumTitle, genre*}', '{*artistId, albumId*}', and '{*albumId, songId*}' since '*albumTitle*' exists in the set of attributes '{*albumId, albumTitle, genre*}'. Whereas the functional dependency '*albumId* → *releaseDate*' will not add an attribute at all since there isn't any attribute set that contains both attributes '*albumId*' and '*releaseDate*'. This process will be recursively performed with all the functional dependencies in the closure for the transformed ORA-SS schema until there is no change to the sets of attributes. The recursively joined sets of attributes are shown below.

{{*artistId, artistName*}, {*releaseDate*}, {*artistId, artistName, albumId, albumTitle, genre*}, {*producerId, producerName, company*}, {*artistId, artistName, albumId, albumTitle, genre, songId, songTitle*}, {*artistId, artistName, albumId, albumTitle, genre*}, *and* {*artistId, artistName, producerId, producerName, company*} }

The '*isLossless*' function compares the set of attributes shown above with all the attributes of the ORA-SS schema in Section 2.1 Figure 1(a) and returns 0 since the set of attributes shown above is not the same as the attributes of the ORA-SS schema which is {*artistId, artistName, albumId, albumTitle, genre, releaseDate, producerId, producerName, company, songId, songTitle*}. This means that the schema in Figure 2(a) does not satisfy the lossless property. It can be seen that the transformation of attribute '*releaseDate*' and object class '*producer*' do cause data loss or spurious data creation.

The correctness criteria for the result of the normalization algorithms of semistructured data is defined in the Z language and contains the verifications of dependency preserving and lossless properties. These declarative representations can be used as a guide to detect areas of concern in already defined normalization algorithms.

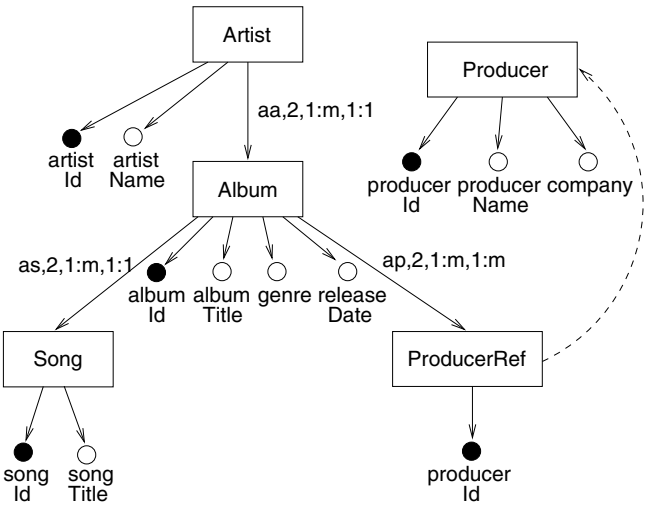


Figure 4. A correctly normalized ORA-SS schema diagram

The result of applying declaratively defined verifications for the dependency preserving and lossless properties can show which part of the transformation causes the dependency loss, data loss, or creation of spurious data. Using this information, the transformation can be corrected. We show this by taking the ORA-SS schema in Section 2.1, Figure 1(a) and transforming it to the schema shown in Figure 4.

4. Conclusion

With the rapid growth of semistructured data usage, the need to design good semistructured schemas has become more and more critical. Normalization is a crucial process that analyzes and transforms the schema of a semistructured data model to minimize redundancies and improve efficiency. During the normalization process, it is essential to ensure the semantic equivalence of the transformed schema with respect to its original form. In this paper, we presented a set of correctness criteria for semistructured data normalization. We proposed two correctness criteria using dependency preserving and lossless properties specific to the semistructured data context. We defined these two criteria in terms the ORA-SS data modeling language, using the Z formal specification language. These correctness criteria can be used to prove the correctness of different normalization algorithms for semistructured data. Furthermore, the formal description of these correctness criteria in Z provides

a solid foundation for automated verification of the normalization algorithms defined for semistructured database systems.

In future work, we plan to apply the correctness criteria proposed in this paper to verify the normalization algorithms that have already been defined for semistructured data, and further to provide automated verification support for the verification process. By doing this we will be able to characterize the normalization algorithms, which will provide a better understanding of the differences between those proposed algorithms. In addition, the correctness criteria can be extended to verify the results of algorithms for view creation and other algorithms that manipulate schemas for semistructured data.

References

[1] C. Anutariya, V. Wuwongse, E. Nantajeewarawat, and K. Akama. Towards a foundation for xml document databases. In K. Bauknecht, S. K. Madria, and G. Pernul, editors, *Electronic Commerce and Web Technologies, First International Conference, EC-Web 2000*, volume 1875 of *Lecture Notes in Computer Science*, pages 324–333. Springer, 2000.

[2] M. Arenas and L. Libkin. A normal form for XML documents. *ACM Trans. Database Syst.*, 29(1):195–232, 2004.

[3] N. Bidoit, S. Cerrito, and V. Thion. A first step towards modeling semistructured data in hybrid multimodal logic. *Journal of Applied Non-Classical Logics*, 14(4):447–475, 2004.

[4] D. Calvanese, G. D. Giacomo, and M. Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.

[5] G. Conforti and G. Ghelli. Spatial tree logics to reason about semistructured data. In S. Flesca, S. Greco, D. Saccà, and E. Zumpano, editors, *Proceedings of the Eleventh Italian Symposium on Advanced Database Systems, SEBD 2003, Cetraro (CS), Italy, June 24-27, 2003*, pages 37–48. Rubettino Editore, 2003.

[6] G. Dobbie, X. Wu, T. Ling, and M. Lee. ORA-SS: Object-relationship-attribute model for semistructured data. Technical Report TR 21/00, School of Computing, National University of Singapore, 2001.

[7] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Addison-Wesley, 4th edition, 2004.

[8] D. W. Embley and W. Y. Mok. Developing XML documents with guaranteed "good" properties. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 426–441, London, UK, 2001. Springer-Verlag.

[9] E. R. Harold and W. S. Means. *XML in a Nutshell*. O'Reilly, Sebastopol, 3rd edition, 2004.

[10] S. U.-J. Lee, J. Sun, G. Dobbie, and Y. F. Li. A Z Approach in Validating ORA-SS Data Models. In *SVV '05: Proceedings of the Third International Workshop on Software Verification and Validation*, volume 157, pages 95–109, October 2005.

[11] T. Ling, M. Lee, and G. Dobbie. Applications of ORA-SS: An Object-Relationship-Attribute data model for semistructured data. In *IIWAS '01: Proceedings of 3rd International Conference on Information Integration and Web-based Applications and Serives*, pages 17–28, Linz, Austria, 2001.

[12] T. W. Ling, M. L. Lee, and G. Dobbie. *Semistructured Database Design*. Springer-Verlag, 2005.

[13] M. Saaltink. The Z/EVES system. In *ZUM '97: Proceedings of the 10th International Conference of Z Users on The Z Formal Specification Notation*, pages 72–85, London, UK, 1997. Springer-Verlag.

[14] M. Srivas, H. Rueß, and D. Cyrluk. Hardware verification using PVS. In T. Kropf, editor, *Formal Hardware Verification: Methods and Systems in Comparison*, volume 1287 of *Lecture Notes in Computer Science*, pages 156–205. Springer-Verlag, 1997.

[15] H. S. Thompson, C. Sperberg-McQueen, N. Mendelsohn, D. Beech, and M. Maloney. XML schema 1.1 part 1: Structures. http://www.w3.org/TR/xmlschema11-1/.

[16] J. Vitt and J. Hooman. Assertional specification and verification using pvs of the steam boiler control system. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler Control*, volume 1165, pages 453–472. Springer-Verlag, 1996.

[17] J. Woodcock and J. Davies. *Using Z: Specification, Refinement, and Proof*. Prentice-Hall, 1996.

[18] X. Wu, T. W. Ling, M. L. Lee, and G. Dobbie. Designing Semistructured Databases Using ORA-SS Model. In *WISE '01: Proceedings of the Second International Conference on Web Information Systems Engineering (WISE'01) Volume 1*, pages 171–182, Kyoto, Japan, 2001. IEEE Computer Society.

[19] X. Wu, T. W. Ling, M. L. Lee, S. Y. Lee, and G. Dobbie. NF-SS: A normal form for semistructured schemata. In *In Proceedings of International Workshop on Data Semantics in Web Information Systems (DASWIS-2001)*, pages 292–305, Yokohama, Japan, November 2001. Springer-Verlag.