A Z Approach in Validating ORA-SS Data Models

Scott Uk-Jin Lee [1] Jing Sun [2] Gillian Dobbie [3]

Department of Computer Science
The University of Auckland
Auckland, New Zealand

Yuan Fang Li [4]

School of Computing
National University of Singapore
Singapore, Republic of Singapore

Abstract

The rapid growth of the World Wide Web has resulted in more data being accessed over the Internet. In turn there is an increase in the use of semistructured data, which plays a crucial role in many web applications particularly with the introduction of XML and its related technologies. This increase in use makes the design of good semistructured data structures essential. The Object Relationship Attribute model for Semistructured data (ORA-SS) is a graphical notation for designing and representing semistructured data. In this paper, we demonstrate an approach to formally validate the ORA-SS data models in order to enhance the correctness of semistructured data design. A mathematical semantics for the ORA-SS notation is defined using the Z formal language, and further validation processes are carried out to check the correctness of the semistructured data models at both the schema and instance levels.

Key words: Semistructured data, ORA-SS data model, Formal specification and verification, Z specification language.

1 Introduction

The rapid growth of the World Wide Web and its technologies has resulted in enormous amounts of data being used over the Internet by Web Services

[1] Email: slee180@ec.auckland.ac.nz
[2] Email: j.sun@cs.auckland.ac.nz
[3] Email: gill@cs.auckland.ac.nz
[4] Email: liyf@comp.nus.edu.sg

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

and other Web-based applications. Many of the applications use semistructured data, which is commonly represented by eXtensible Markup Language (XML) [2] and its related technologies. With the growth in use of semistructured data, the design of good semistructured data structures is essential, particularly if the data is stored in a database [10]. In order for the design and use of semistructured data to be effective and efficient, it is very important to have a good schema definition for semistructured data. The Object Relationship Attribute model for Semistructured data (ORA-SS) data modeling langauge [5] was introduced for this purpose. For ORA-SS to be utilised widely, it is essential to define a formal mathematical semantics for ORA-SS since the current representation of ORA-SS is restricted to a diagrammatic notation and semantics written in English. The benefits of having such a formal semantics include:

- remove ambiguity that may arise from a diagrammatic representation,
- enable the use of ORA-SS in other applications and tools, and
- reveal inconsistencies in a design at the schema and instance levels.

Inconsistencies at the schema level arise if a customized ORA-SS schema model does not conform to the ORA-SS notation. Inconsistencies at the instance level arise if an XML document is not consistent with its schema. For example, an inconsistency that might arise at the schema level is the specification of a ternary relationship between only two object classes. An inconsistency that might arise at the instance level is a many to many relationship between elements when a one to many relationship is specified in the schema. These two aspects of validation are essential in the semistructured data design process. Thus, the provision of formal semantics and reasoning support for validating ORA-SS semistructured data modeling is very beneficial. Furthermore, this validation also improves the quality of applications utilizing semistructured data. Traditional validation of semistructured data is limited to syntax checking only, e.g., XML Schema or DTD. However, deep semantic checking and reasoning adds to the validation of semistructured data design. The quality of the software system will surely improve when these validation tasks are available for applications which use semistructured data because methods of ensuring correctness have expanded from plain syntax checking to semantics checking.

In this paper, we demonstrate an approach to formally validate the ORA-SS data models in order to enhance the correctness of semistructured data design. Firstly, a mathematical semantics for the ORA-SS diagrammatic notation is defined using the Z [8] formal language. Secondly we show how the mathematical semantics is used in both the schema level and instance level validation. There is related work using multimodal logic [1] and spatial tree logic [4] to present and reason about semistructured data. And a description logic representation of XML documents can be found in [3]. While this work has helped us define the semantics of ORA-SS, there is no promise of

automated reasoning being applied to any of it at the moment.

The rest of the paper is organized as follows. Section 2 presents the background knowledge on ORA-SS notation and the Z formal language. Section 3 presents a formal semantics of the ORA-SS language in Z first-order logic. In Section 4, we demonstrate the reasoning process of validating semistructured data models on both an ORA-SS schema diagram and its XML instance. Section 5 concludes the paper and describes future work.

2 Background

2.1 ORA-SS data modeling language

The Object Relationship Attribute model for Semistructured data (ORA-SS) data modeling language [5,6] consists of four basic concepts: object class, relationship type, attribute and reference. It represents these concepts through four diagrams: schema diagram, instance diagram, functional dependency diagram and inheritance diagram.

- An object class is like an entity type in an ER diagram, a class in an object-oriented diagram or an element in an XML document. The object classes are represented as labelled rectangles in an ORA-SS diagram.
- A relationship type represents a nesting relationship among object classes. It is represented optionally with a labelled diamond and can be described by `name`, `n`, `p` and `c`. The `name` denotes the name of relationship type, integer `n` indicates degree of relationship type, `p` represents participation constraint of parent object class in relationship type and `c` represents participation constraint of child object class in relationship type. The constraints are represented using `min:max` notations with abbreviated symbols.
- Attributes represent properties and are denoted by labelled circles. An attribute can be a key attribute which has a unique value and represented as a filled circle. An attribute can be a property of an object or a property of a relationship. An attribute of an object class has no label on its incoming edge and an attribute of a relationship has the name of the associated relationship type on its incoming edge.
- An object class can reference another object class to model recursive and symmetric relationships, or to reduce redundancy especially for many-to-many relationships. It is represented by a labelled dashed edge. Disjunction of objects and attributes is another thing that can be represented.

For example, Figure 1 presents an ORA-SS schema that represents the structure of a particular semistructured data. This schema diagram shows that there is a relationship between the '`course`' object class and the '`student`' object class, and this relationship has a single-valued attribute '`mark`'. The object class '`course`' has an identifier '`code`', with single-valued attribute '`title`' and multi-valued attribute '`ANY`'. Object class '`student`' has iden-

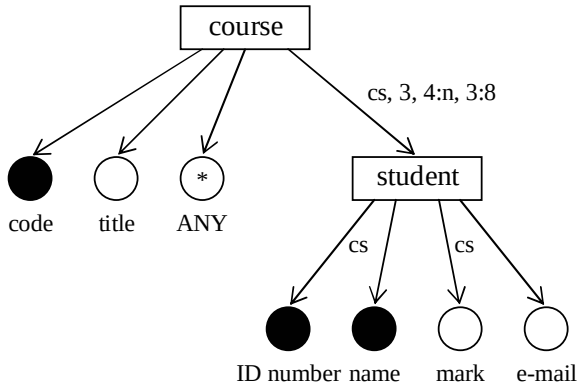

Fig. 1. An example of an invalid ORA-SS schema diagram.

tifier '`ID number`', and single valued attributes '`name`' and '`email`'. However, the above schema diagram is syntactically correct but there are three semantic errors. The degree of relationship '`cs`' is 3, representing a ternary relationship where it actually is a binary relationship since object '`course`' is not related to any other objects besides '`students`'. Another semantic error is having two primary keys for the object class '`student`'. There are two attributes selected as primary key where there should only be one primary key for each object class. The third semantic error in this schema is that the primary key '`ID number`' is represented as an attribute of the relationship '`cs`' where it really is an attribute of an object '`student`'. A validation process should pick up this kind of errors in the design.

2.2 Z formal specification langauge

Z [9] is a formal specification language originally developed at the Programming Research Group at Oxford University and it is probably the most popular formal specification language currently available. It is based on set theory and first-order predicate logic. Z is a declarative language and it has a number of language constructs including given type, abbreviation type, axiomatic definition, state and operation schema definitions. It has been widely used for providing formal semantics and verifications in various application domains. Z/EVES [7] is an interactive system for composing, checking, and analyzing Z specifications. In particular, it supports general theorem proving of Z specifications.

3 Formal semantics of ORA-SS data modeling langauge

3.1 Basic types

Initially basic types used in the ORA-SS data modeling language must be identified and defined prior to constructing the formal representation.

$$[OBJCLASS, OBJECT, ATTRIBUTE, ATTVALUE]$$

The object types and attribute types defined above represent the set of object classes, object instances, attributes and attribute values respectively in the ORA-SS language.

3.2 Relationship type

Relationship type is defined as a function with a set of object classes as its domain and a sequence of set of object classes as its range. The predicate of the function uses a recursive definition and describes that object classes can be related to other object classes as well as to other relationships.

$$
\begin{array}{|l}
relationship : \mathbb{P}\, OBJCLASS \rightarrow \mathrm{seq}_1\,(\mathbb{P}\, OBJCLASS) \\
\hline
\forall\, objclses : \mathbb{P}\, OBJCLASS;\ seqobjclses : \mathrm{seq}_1\,(\mathbb{P}\, OBJCLASS) \\
\quad \bullet \{objclses \mapsto seqobjclses\} \subset relationship \\
\quad \Leftrightarrow (\forall\, objclses0 : \mathrm{ran}\, seqobjclses \bullet objclses \cap objclses0 = \emptyset) \\
\qquad \wedge\, (\#seqobjclses = 1 \\
\qquad\quad \vee\, (\#seqobjclses \geq 2 \\
\qquad\qquad \Rightarrow \{head\, seqobjclses \mapsto tail\, seqobjclses\} \subset relationship))
\end{array}
$$

In an ORA-SS schema diagram, there are two types of relationship, i.e., a normal relationship where the child participant is a single object class; and a disjunctive relationship where the child participant is a set of disjunctive object classes. The above definition includes both cases. It defines a relationship as a function where the first argument represents the child object classes in a relationship and the second argument represents a sequence of set of object classes which refers to either a parent object class or another sub-relationship. Note that the '$\mathbb{P}\, OBJCLASS$' denotes both the normal relationship case where the child is a singleton object class set, and the disjunctive relationship case where the child is a set of disjunctive object classes.

The first predicate of the function defines that the child object classes should not intersect with any of the object classes at the parent level. This is to prevent cyclic definitions in the relationship structure. The second predicate states that child object classes can be connected to either a singleton sequence of object classes which forms a binary relationship, or with a sequence that has more than two elements which represents a relationship of degree 3 or more. It specifies that when the cardinality of the object classes sequence is greater or equal to two, the head of the sequence and its tail forms another sub-relationship type.

3.3 Degree of a relationship type

Every relationship in an ORA-SS schema diagram has its associated degree represented as a natural number.

$$
\begin{array}{|l}
degree : relationship \rightarrow \mathbb{N}_1 \\
\hline
\forall\, rel0 : relationship \bullet degree\, rel0 = \#(second\, rel0) + 1
\end{array}
$$

The above definition defines a degree as a function where the first argument represents the relationship and the second argument represents the natural number which refers to the value of the degree of the relationship. The predicate of the function defines that the degree of any relationship is 1 added to the cardinality of the sequence of set of object classes which represents either a parent object class or a sub-relationship. It basically specifies the degree of a relationship is the number of object classes involved in the relationship.

3.4 Instances of object class and attribute

In the ORA-SS data model, an object class has instances which are objects.

$$
\begin{array}{|l}
hasObjInstance : OBJCLASS \rightarrow \mathbb{P}\, OBJECT \\
\hline
\forall\, object1, object2 : OBJCLASS \bullet object1 \neq object2 \\
\quad \Rightarrow hasObjInstance\ object1 \cap hasObjInstance\ object2 = \emptyset
\end{array}
$$

The above definition defines object classes having instances as a function. In this function, the first argument represents an object class and the second argument represents a set of objects which refers to all the instances of the object class. The predicate of the function defines that any two different object classes should have different object instances. This specifies that an object cannot be an instance of multiple object classes. As an object class has instances, attributes also have values. A similar definition can be provided.

3.5 Instances of a relationship type

In the ORA-SS data model, a relationship type also has its instances which represents the participation instances from their corresponding object classes in the relationship.

$$
\begin{array}{|l}
hasRelInstance : relationship \rightarrow (OBJECT \leftrightarrow \mathrm{seq}_1\ OBJECT) \\
\hline
\forall\, rel0 : relationship;\ relinsts0 : OBJECT \leftrightarrow \mathrm{seq}_1\ OBJECT \\
\bullet\ relinsts0 = hasRelInstance\ rel0 \\
\Leftrightarrow (\forall\, relinst : relinsts0 \\
\quad \bullet\ \#(second\ relinst) + 1 = degree\ rel0 \\
\quad \wedge\ (\exists_1\ objcls : first\ rel0 \bullet \\
\qquad \mathrm{dom}(relinsts0 \rhd \{second\ relinst\}) \subseteq hasObjInstance\ objcls) \\
\quad \wedge\ ((\#(second\ relinst) = 1 \\
\qquad \Rightarrow \#(second\ rel0) = 1 \\
\qquad\quad \wedge\ (\exists_1\ objcls : head(second\ rel0) \\
\qquad\qquad \bullet\ head(second\ relinst) \in hasObjInstance\ objcls)) \\
\quad \vee\ (\#(second\ relinst) \geq 2 \\
\qquad \Rightarrow (head(second\ relinst), tail(second\ relinst)) \\
\qquad\quad \in hasRelInstance(head(second\ rel0) \mapsto tail(second\ rel0))))) \\
\forall\, rel1, rel2 : relationship \mid rel1 \neq rel2 \\
\quad \bullet\ hasRelInstance\ rel1 \cap hasRelInstance\ rel2 = \emptyset
\end{array}
$$

The above definition defines relationship having instances as a function where the first argument represents a relationship and the second argument represents the instance of the relationship. An instance of the relationship is represented as an object related to a sequence of objects that conforms to the relationship definition.

The first predicate of the function defines that the degree of a relationship instance should be the same as the degree of the relationship type. The second predicate defines that child object instance should be an instance of the associated selected child object classes in the relationship. Also the predicate defines that only the objects of a single object class is related to a parent object or sub-relationship instance in the case of a disjunctive relationship. The third predicate consists of two cases. If the degree of the relationship is binary, the parent object instance should be an instance of the parent object class. If the degree of the relationship is ternary or more, the second part of the predicate recursively defines that the sub-relationship instance sequence is an instance of the sub-relationship type. Finally, the last predicate defines that any two relationship types should have their own disjoint set of instances. This is to specify that a relationship instance cannot be an instance of multiple relationship types.

3.6 Participation constraints on objects in a relationship type

Every relationship type in an ORA-SS schema diagram has its associated constraints on its participating objects which is represented by the '`min:max`' notation. It constrains the number of child objects that a parent object can relate to and vice versa.

$$
\begin{array}{|l}
parentConstraints : relationship \rightarrow (MULTIPLICITY \times MULTIPLICITY) \\
\hline
\forall\, rel0 : relationship;\ card : (MULTIPLICITY \times MULTIPLICITY) \\
\quad \bullet\ parentConstraints\ rel0 = card \\
\quad \Leftrightarrow (\#(second\ rel0) < 2 \\
\qquad \Rightarrow (\forall\, obj : OBJECT \\
\qquad\quad \mid (\exists_1\, objcls : head\ (second\ rel0) \bullet obj \in hasObjInstance\ objcls) \\
\qquad\quad \bullet\ card.1 \leq \#((hasRelInstance\ rel0) \rhd \{\langle obj \rangle\}) \leq card.2)) \\
\quad \wedge (\#(second\ rel0) \geq 2 \\
\qquad \Rightarrow (\forall\, subrelinst : hasRelInstance\ (head\ (second\ rel0) \\
\qquad\qquad\qquad\qquad\qquad\qquad \mapsto tail\ (second\ rel0)) \\
\qquad\quad \bullet\ card.1 \leq \#((hasRelInstance\ rel0) \\
\qquad\qquad \rhd \{(\langle first\ subrelinst \rangle \frown second\ subrelinst)\}) \leq card.2))
\end{array}
$$

The above definition defines parent constraints as a function where the first argument represents a relationship and the second argument represents a cartesian product of multiplicity which refers to a '`min:max`' pair. The predicate of the function defines that the number of relationship instances in which each object of the parent object class or each relationship instance of the sub-relationship type should be within the multiplicities defined in the relationship.

It specifies that the parent constraint sets the boundaries for the number of child objects that a single parent object or sub-relationship instance can have. The child constraints of the relationship can be defined in a similar way.

3.7 Object class, attribute pairs and their instances

In ORA-SS schema diagrams, an object can have a set of attributes.

$$hasObjectAttribute : OBJCLASS \rightarrow \mathbb{P}\, ATTRIBUTE$$

The association between attributes and an object class also has a set of instances, which is defined in the definition '`hasObjAttrInstance`'.

$$hasObjAttrInstance : hasObjectAttribute \rightarrow (OBJECT \leftrightarrow \mathbb{P}\, ATTVALUE)$$

$$\begin{aligned}
&\forall\, objattr : hasObjectAttribute;\ objattrinsts : (OBJECT \leftrightarrow \mathbb{P}\, ATTVALUE) \\
&\quad \bullet\ hasObjAttrInstance\ objattr = objattrinsts \\
&\quad \Leftrightarrow (\forall\, objattrinst0 : objattrinsts \\
&\qquad \bullet\ (first\ objattrinst0) \in hasObjInstance(first\ objattr) \\
&\qquad \wedge\ \#(second\ objattrinst0) = \#(second\ objattr) \\
&\qquad \wedge\ (\forall\, attrVal : second\ objattrinst0 \\
&\qquad\quad \bullet\ (\exists_1\, attr : second\ objattr \bullet attrVal \in hasAttValue\ attr)) \\
&\qquad \wedge\ (\forall\, attr : second\ objattr \bullet (\exists_1\, attrVal : second\ objattrinst0 \\
&\qquad\quad \bullet\ attrVal \in hasAttValue\ attr)))
\end{aligned}$$

The first argument represents 'hasObjectAttribute' function and the second argument represents a set of pair of an object and set of attribute values. The first two predicates define that for all the objects and attribute value sets, the object should belong to the object class in the '`hasObjectAttribute`' function and the cardinality of the attribute value set should be the same as the cardinality of the attribute in the '`hasObjectAttribute`' function. The third and fourth predicates define that each attribute value should belong to one and only one corresponding attribute specified in '`hasObjectAttribute`' function and vice versa. Attributes can belong to relationships and relationship attributes have their own instances. The relationship attributes and their instance can be defined similarly.

3.8 Candidate key and primary key of an object class

An object can have an attribute or set of attributes that have a unique value for each instance of an object class called a candidate key. There are two types of candidate key, i.e., a candidate key where the key is a single attribute; and a composite candidate key where the key is a set of attributes. The following definition includes both cases.

$$
\begin{array}{|l}
hasObjectCandidateKey : OBJCLASS \leftrightarrow \mathbb{P}\, ATTRIBUTE \\
\hline
\forall\, object0 : OBJCLASS;\ key0 : \mathbb{P}\, ATTRIBUTE \\
\bullet\ (object0, key0) \in hasObjectCandidateKey \\
\Leftrightarrow key0 \subseteq hasObjectAttribute\ object0 \\
\quad \wedge\ (\forall\, keyValue1, keyValue2 : \mathbb{P}\, ATTVALUE; \\
\qquad objInst1, objInst2 : hasObjInstance\ object0 \\
\qquad \mid (\forall\, attrVals1 : \mathrm{ran}(\{objInst1\} \\
\qquad\quad \lhd hasObjAttrInstance(object0, (hasObjectAttribute\ object0))) \\
\qquad\quad \bullet\ keyValue1 \subseteq attrVals1) \\
\qquad \wedge\ (\forall\, attrVals2 : \mathrm{ran}(\{objInst2\} \\
\qquad\quad \lhd hasObjAttrInstance(object0, (hasObjectAttribute\ object0))) \\
\qquad\quad \bullet\ keyValue2 \subseteq attrVals2) \\
\qquad \bullet\ ((keyValue1 \neq keyValue2 \Rightarrow objInst1 \neq objInst2) \\
\qquad\quad \wedge\ (keyValue1 = keyValue2 \Rightarrow objInst1 = objInst2)))
\end{array}
$$

It defines the object having a candidate key as a relationship where object classes are related to the set of attributes which refer to all the candidate keys that belong to the object. Note that the '$\mathbb{P}$ `ATTRIBUTE`' denotes both a candidate key and a composite candidate key. The first predicate of the function defines that candidate keys belong to the set of attributes that the object has. The second predicate of the function defines two facts. It states that two objects are different when values of the candidate key for each object are different; and two objects are the same when values of the candidate key for each object are the same, where the values of the candidate keys belongs to the set of attribute values of the object attributes. This specifies that the value of candidate key for each object of an object class should uniquely identify an object instance.

$$
\begin{array}{|l}
hasObjectPrimaryKey : OBJCLASS \rightarrow \mathbb{P}\, ATTRIBUTE \\
\hline
hasObjectPrimaryKey \subseteq hasObjectCandidateKey
\end{array}
$$

In ORA-SS schema diagrams, an object class has a primary key which is selected from the set of candidate keys. The '`hasObjectPrimaryKey`' is defined as a total function type which indicates each object class can relate to one and only one set of attributes as its primary key.

3.9 Cardinality of attribute values associated with an object

As a relationship type has its associated participation constraints, attributes also has cardinality constraints associated with an object instance. It constrains the number of attribute values that an object can have.

$$
\begin{array}{|l}
hasAttCardinality : ATTRIBUTE \rightarrow (MULTIPLICITY \times MULTIPLICITY) \\
\hline
\forall\, attr0 : ATTRIBUTE;\ card : MULTIPLICITY \times MULTIPLICITY \\
\bullet\ hasAttCardinality\ attr0 = card \\
\Leftrightarrow (\forall\, obj : OBJCLASS \mid attr0 \in hasObjectAttribute\ obj \\
\quad \bullet\ (\forall\, objinst : hasObjInstance\ obj \\
\quad\quad \bullet\ card.1 \leq \#(\{objinst\} \\
\quad\quad\quad \lhd hasObjAttrInstance(obj, (hasObjectAttribute\ obj))) \\
\quad\quad \leq card.2)) \\
\vee\ (\forall\, rel : relationship \mid attr0 \in hasRelationshipAttribute\ rel \\
\quad \bullet\ (\forall\, relinst : hasRelInstance\ rel \\
\quad\quad \bullet\ card.1 \leq \#(\{relinst\} \\
\quad\quad\quad \lhd hasRelAttrInstance(rel, (hasRelationshipAttribute\ rel))) \\
\quad\quad \leq card.2))
\end{array}
$$

The above definition states the cardinality constraints of attributes as a function where first argument represents attribute and the second argument represent a cartesian product of multiplicity which refers to the '`min:max`' form. The predicate of the function defines that the number of attribute values for each object or relationship instance should be within the multiplicities specified on the attribute. In this section, we presented some of the formal semantics of the ORA-SS language constructs using the Z first-order logic. Due to the space limit, not all the semantic definitions are presented here. Other ORA-SS constructs can be defined in a similar manner.

4 Validating semistructured data

As we mentioned earlier, a major concern in designing a good semistructured data structure for a particular application is to ensure there are no possible inconsistencies in either the ORA-SS schema diagram or the XML instance. Having defined a formal semantics of ORA-SS in Z, we can present and validate any ORA-SS schema diagram and its XML instances.

4.1 Schema diagram validation

Schema validation involves checking whether a customized ORA-SS schema diagram is consistent according to the semantics of the ORA-SS language. Possible guidelines for validating an ORA-SS schema diagram are as follows.

- In a relationship type, the child object class must be either related to another parent object class to form a binary relationship or related to another sub-relationship type to form a relationship type of degree 3 or more.
- The degree of a binary relationship is 2, ternary is 3 and n-nary is n.
- In a disjunctive relationship type, the child participants is a set of disjunctive object classes.

- A composite attribute or disjunctive attribute has an attribute that is related to two or more sub-attributes.
- A candidate key of an object class is selected from the set of attributes of the object class.
- A composite key is selected from 2 or more attributes of an object class.
- There can only be one primary key per object class and it can be either a candidate key or a composite candidate key.
- Relationship attributes have to relate to an existing relationship.
- An object class can reference one object class only, but an object class can be referenced by multiple object classes.

The above are some of the criteria for validating a schema diagram against the ORA-SS notation. As we can see from the previous section, most of these guidelines have already been encoded into the Z semantics of ORA-SS. Thus when we represent a particular ORA-SS schema model in Z, we can validate the correctness of the schema diagram against the ORA-SS Z semantics. For example, we can represent the schema diagram in Figure 1 and validate it as follows.

$$
\begin{array}{|l}
Course, Student : OBJCLASS \\
Code, Title, Any, IDNumber, Name, Mark, EMail : ATTRIBUTE \\
cs : relationship \\
\hline
hasObjectAttribute\ Course = \{Code, Title, Any\} \\
hasObjectAttribute\ Student = \{Name, EMail\} \\
cs = \{Student\} \mapsto \langle\{Course\}\rangle \\
degree\ cs = 3 \\
parentConstraints\ cs = (4, many) \\
childConstraints\ cs = (3, 8) \\
hasRelationshipAttribute\ cs = \{IDNumber, Mark\} \\
hasObjectPrimaryKey\ Course = \{Code\} \\
hasObjectPrimaryKey\ Student = \{IDNumber\} \\
hasObjectPrimaryKey\ Student = \{Name\}
\end{array}
$$

- **Validating the degree of the `cs` relationship**:

$$
\begin{aligned}
&degree\ cs = degree(\{Student\} \mapsto \langle\{Course\}\rangle) \\
&= \#(second(\{Student\} \mapsto \langle\{Course\}\rangle)) + 1 \\
&= \#(\langle\{Course\}\rangle) + 1 = 2 \neq 3 \\
&\Rightarrow false
\end{aligned}
$$

  Thus the definition of $degree\ cs = 3$ is invalid.

- **Validating the primary key of the `Student` object class**:

$$
\begin{aligned}
&hasObjectPrimaryKey\ Student = \{IDNumber\} \\
&\Rightarrow Student \mapsto \{IDNumber\} \in hasObjectCandidateKey \\
&\Rightarrow \{IDNumber\} \subseteq hasObjectAttribute\ Student \\
&\Rightarrow \{IDNumber\} \subseteq \{Name, EMail\} \\
&\Rightarrow false
\end{aligned}
$$

Thus the attribute '`IDNumber`' can not be a primary key of the '`Student`' object class since it is an attribute of the relationship '`cs`'.

$$
\begin{aligned}
&\mathit{hasObjectPrimaryKey}\ \mathit{Student} = \{\mathit{IDNumber}\} \\
&\land\ \mathit{hasObjectPrimaryKey}\ \mathit{Student} = \{\mathit{Name}\} \\
&\Rightarrow \{\mathit{Student} \mapsto \{\mathit{IDNumber}\}, \mathit{Student} \mapsto \{\mathit{Name}\}\} \subseteq \mathit{hasObjectPrimaryKey} \\
&\land\ \mathit{hasObjectPrimaryKey} \subseteq \mathit{OBJCLASS} \rightarrow \mathbb{P}\ \mathit{ATTRIBUTE} \\
&\Rightarrow \mathit{false}
\end{aligned}
$$

Since '`hasPrimaryKey`' is defined as a function mapping from an object class to a set of attributes as its primary key, one object class in the domain can not be mapped to two different values in the range in a function definition. Thus the attribute '`IDNumber`' and '`Name`' can not both be the primary keys of the object class '`Student`'. Intuitively, we chose the '`IDNumber`' as the key since the '`Name`' may not uniquely identify a student object.

After revealing all the errors we can correct the ORA-SS schema example in Figure 1 by removing the '`name`' attribute in the student as a primary key.

4.2 XML instance validation

The XML instance validation is defined to check whether there are any possible inconsistencies in a semistructured data instance, where an XML document should be consistent with regard to the designated ORA-SS schema diagram. Possible guidelines for validating an XML instance are as follow.

- Relationship instances must conform to the parent participation constraints, e.g., the number of child objects related to a single parent object or relationship instance should be consistent with the parent participation constraints; and the number of parent objects or relationship instances that a single child object relates to should be consistent with the child participation constraints.
- In a disjunctive relationship, only one object class can be selected from the disjunctive object class set and associated to a particular parent instance.
- For a candidate key (single or composite), its value should uniquely identify the object that this key attribute belongs to.
- Each object can have one and only one primary key.
- All attributes have their own cardinality and the number of attributes that belong to an object should be limited by the minimum and maximum cardinality values of the attribute.
- For a set of disjunctive attributes, only one of the attribute choices can be selected and associated to an object instance.

These are some of the criteria of instance level validation. Given an XML instance file, we should be able to check the consistency of the content in the document against its ORA-SS schema definitions. For example, the following is an XML document that should conform to the corrected ORA-SS schema

definition in Figure 1 [5].

```
<Example>
  <Course code = "CS101">
        <title>Principles of Programming</title>
    <Student IDNo = "1111111">
        <Name>Scott Lee</Name>
        <Mark>A</Mark>
        <EMail>slee180@auckland.ac.nz</EMail>
    </Student>
    <Student IDNo = "2222222">
         ...
    </Student>
    ...
  </Course>
  <Course code = "CS105">
        <title>Principles of Computer Science</title>
    <Student IDNo = "1111111">
    ...
    </Student>
    ...
  </Course>
</Example>
```

We can traslate the above XML instance into our ORA-SS Z semantics [6]. The following shows some of the validation steps regarding the above XML instance.

- **Validating participation constraints of the relationship type `cs`:**

$$
\begin{aligned}
& parentConstraints\ cs = (4, many) \\
& \Rightarrow \forall\ courseinst : hasObjInstance\ Course \\
& \qquad \bullet\ 4 \leq \#((hasRelInstance\ cs) \rhd \{\langle courseinst \rangle\}) \leq many \\
& \land \exists_1\ Course8 : hasObjInstance\ Course \\
& \qquad \bullet\ \#((hasRelInstance\ cs) \rhd \{\langle Course8 \rangle\}) = 3 \\
& \Rightarrow 4 \leq 3 \Rightarrow false \\
& \land \exists_1\ Course9 : hasObjInstance\ Course \\
& \qquad \bullet\ \#((hasRelInstance\ cs) \rhd \{\langle Course9 \rangle\}) = 3 \\
& \Rightarrow 4 \leq 3 \Rightarrow false
\end{aligned}
$$

From the above, we can conclude that '`Course8`' and '`Course9`' does not satisfy the parent participation constraint of a minimum of 4 students per course. Since we know that the course code is a primary key of a course object from the '$hasObjPrimaryKey\ Course = \{Code\}$' definition, we can trace that course '`CS334`' and '`CS340`' in the XML document fails to satisfy the participation constraint defined in the ORA-SS schema diagram. Similarly, we can check the child participation constraint of the relationship '`cs`'.

[5] Due to the space limit of the paper, only a small part of the XML definitions is listed here. A complete XML file of the above example can be found at '`www.cs.auckland.ac.nz/~jingsun/example.xml`'.

[6] A complete Z representation of the XML instance example above can be found at '`www.cs.auckland.ac.nz/~jingsun/example-z.ps`'.

This time, all the child participation constraints are met, as each student object had 3 to 8 course objects in its relationship instance.

- **Validating the primary key values of a `Student` object**:

$$
\begin{aligned}
& hasObjectPrimaryKey\ Student = \{IDNumber\} \\
& \Rightarrow (Student, \{IDNumber\}) \in hasObjectCandidateKey \\
& \Rightarrow \forall\, id_1, id_2 : hasAttValue\ IDNumber;\ stu_1, stu_2 : hasObjInstance\ Student \\
& \quad \mid ((\forall\, attrVals_1 : \mathrm{ran}(\{stu_1\} \\
& \qquad \lhd hasObjAttrInstance\ (Student \mapsto \{IDNumber, Name, EMail\})) \\
& \qquad \bullet\ id_1 \in attrVals_1) \\
& \quad \land\ (\forall\, attrVals_2 : \mathrm{ran}(\{stu_2\} \\
& \qquad \lhd hasObjAttrInstance\ (Student \mapsto \{IDNumber, Name, EMail\})) \\
& \qquad \bullet\ id_2 \in attrVals_2)) \\
& \quad \bullet\ (id_1 \neq id_2 \Rightarrow stu_1 \neq stu_2) \land (id_1 = id_2 \Rightarrow stu_1 = stu_2) \\
& \land\ \exists (Student3 \mapsto \{No3333333, AnaCole, acol003\}), \\
& \quad (Student6 \mapsto \{No3333333, SarahChan, scha077\}) \\
& \quad : hasObjAttrInstance\ (Student \mapsto \{IDNumber, Name, EMail\}) \\
& \quad \bullet\ (No3333333 = No3333333 \land Student3 \neq Student6) \\
& \Rightarrow false
\end{aligned}
$$

The above shows that the primary key value of '`No3333333`' in the XML instance does not uniquely represent a student object. In this case, the key value '`No3333333`' is associated with the objects '`student3`' and '`student6`'. This is another inconsistency in the XML document that does not follow the ORA-SS schema definition. Similarly, we can perform a primary key check on the course object instance, which satisfies the defined constraints.

More complicated proof steps can be established for validating large XML documents to check the correctness of a semistructured data design.

5 Conclusion

In this paper, we present an approach to formally validate the consistency of semistructured data design. Our work outlines the following three contributions. Firstly, a formal mathematical semantics for the ORA-SS diagrammatic data modeling notation is defined. This formal semantics is useful in providing a rigorous formal foundation for the ORA-SS language. Furthermore, such a semantics can be adopted by many semistructured data applications which use the ORA-SS data model. Secondly, some guidelines for validating the ORA-SS data models were defined at both the schema diagram level and the XML instance level. These validation guidelines can be used as a template for the applications that implement the validation algorithm of ORA-SS semistructured data. Thirdly, we demonstrate some reasoning steps using the Z ORA-SS semantics in validating customized ORA-SS schema diagrams and XML instances. Proof steps are presented through a simple ORA-SS data model. More complicated proofs can also be constructed for validating large semistructured documents.

In the future, we plan to extend and concentrate our work on the automatic validation of semistructured data in Z. We are in the process of developing a theorem library to support the auto-validation of ORA-SS schema diagrams and their data instances using the Z/EVES theorem prover. By doing so, manual proofs can be avoided. Furthermore, we plan to develop a translation program that automatically transforms an XML instance into its corresponding Z ORA-SS instance representation for machine validation. In addition, we also plan to extend the current Z semantics of the ORA-SS language to model the normalization problems in semistructured data design.

References

[1] Bidoit, N., S. Cerrito and V. Thion, *A First Step Towards Modeling Semistructured Data in Hybrid Multimodal Logic*, Journal of Applied Nonclassical Logic **14** (2004).

[2] Bray, T., J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau, *Extensible Markup Language (XML) 1.0 (Third Edition)*, http://www.w3.org/TR/REC-xml/.

[3] Calvanese, D., G. D. Giacomo and M. Lenzerini, *Representing and Reasoning on XML Documents: A Description Logic Approach*, Journal of Logic and Computation **9** (1999), pp. 295–318.

[4] Conforti, G. and G. Ghelli, *Spatial Tree Logics to reason about Semistructured Data*, in: S. Flesca, S. Greco, D. Saccà and E. Zumpano, editors, *SEBD '03: Proceedings of 11th Italian Symposium on Advanced Database Systems* (2003), pp. 37–48.

[5] Dobbie, G., 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, Singapore (2001).

[6] Ling, T. W., M. L. Lee and G. Dobbie, "Semistructured Database Design," Springer, 2005.

[7] Saaltink, M., *The Z/EVES system*, in: J. P. Bowen, M. G. Hinchey and D. Till, editors, *ZUM'97: Z Formal Specification Notation*, Lecture Notes in Computer Science **1212** (1997), pp. 72–85.

[8] Spivey, J., "The Z Notation: A Reference Manual," International Series in Computer Science, Prentice-Hall, 1992.

[9] Woodcock, J. and J. Davies, "Using Z: Specification, Refinement, and Proof," International Series in Computer Science, Prentice-Hall, 1996.

[10] Wu, X., T. W. Ling, M. L. Lee and G. Dobbie, *Designing Semistructured Databases Using the ORA-SS Model*, in: *WISE '01: Proceedings of 2nd International Conference on Web Information Systems Engineering* (2001).