Validating ORA-SS Data Models using Alloy

Lin Wang, Gillian Dobbie, Jing Sun
Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland, New Zealand
lwan066@ec.auckland.ac.nz
{gill, j.sun}@cs.auckland.ac.nz

Lindsay Groves
School of Mathematics, Statistics
and Computer Science
Victoria University of Wellington
P.O. Box 600
Wellington, New Zealand
lindsay@mcs.vuw.ac.nz

Abstract

Semistructured data is typically represented using XML. However, little semantic information can be captured using XML. Other data models, such as the Object Relationship Attribute data model for Semistructured data (ORA-SS), have been introduced to represent more detailed semantic information. Automatic analysis of the data models would enable us to reveal inconsistencies both at the schema and instance levels of the semistructured data. The aim of this paper is to encode the semantics of the ORA-SS data model in the Alloy formal language and automatically validate the semistructured data design using the Alloy Analyzer. It enables us to check the consistency of an ORA-SS schema and its instances.

Keywords: *Semistructured data, ORA-SS, Modeling language semantics, Formal verification and validation.*

1 Introduction

Semistructured data is used in many application areas such as multimedia data management, biological databases, digital libraries, and data integration. In particular, it has become prevalent with the growth of the Internet. EXtended Markup Language (XML) [4], which is a specialization of the Standard Generalized Markup Language (SGML), has become the likely standard for representing and exchanging semistructured data. To date, there is no widely accepted data model for XML or semistructured data. A number of data modeling languages have been proposed for semistructured data [2, 5, 11, 16], including the Object Relationship Attribute data model for Semistructured data (ORA-SS) [9, 15] which captures the constraints that are necessary for the informal modeling of applications and algorithms [14, 15, 19].

The ORA-SS data model not only reflects the nested structure of semistructured data, but also distinguishes between object classes, relationship types and attributes. The main advantages of ORA-SS over other data models is its ability to express the degree of a n-ary relationship type, and distinguish between the attributes of relationship types and the attributes of object classes. This semantic information is essential, even crucial for semistructured data representation and management.

A primary concern in designing a semistructured data model for a particular application is to reveal any possible inconsistencies at both the schema and instance levels. The former refers to the possible errors in a semistructured data schema model, where a customized ORA-SS schema model should be consistent with respect to a given set of constraints. The latter refers to the possible errors in a semistructured data instance, where an XML document should be consistent with respect to the designed schema model. 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 automated tool support for validating ORA-SS semistructured data modeling is very beneficial.

An important part of software engineering is proving properties of new concepts. In this paper, we present an approach to encode the semantics of the ORA-SS data model into the Alloy formal language [12] and use the Alloy Analyzer to perform automatic consistency checking on both the semistructured schema model and its data instances. We chose Alloy over other formal languages for the following reasons. First, Alloy is a light weight formal language that is easily analyzable. Second, Alloy is based on first-order relational logic, and relationships among elements are an

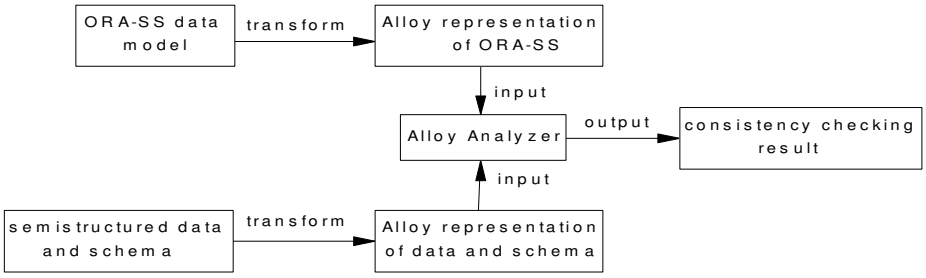

Figure 1. Overall approach to validate semistructured data using Alloy.

important concept in the ORA-SS data model and more generally in semistructured data. More importantly, Alloy has automated tool support - Alloy Analyzer [13]. It is a fully automatic tool built on top of a set of SAT Solvers [20] to simulate and check the specification models written in the Alloy language.

Figure 1 shows our approach in validating semistructured data through Alloy. Firstly, a formal semantics for the ORA-SS data modeling language is defined in Alloy. It provides a precise and rigorous formal basis for representing ORA-SS data schema models and their instances in Alloy. Secondly, with the assistance of the Alloy Analyzer, automated consistency checking on the semistructured data at both the schema and instance levels can be achieved.

There has been other research that provides formal semantics for semistructured data. For example, Calvanese et al. [6] presents a formalization of XML DTD and documents in terms of an expressive description logic. Anutariya et al. [1] formalizes XML DTD and documents in XML declarative description, a theoretical framework developed from declarative description theory. Conforti and Ghelli [7] introduces spatial tree logics, a language based on ambient logic, as formalisms for semistructured data. More recently, Bidoit et al. [3] applies hybrid multimodal logic to formalize the semistructured data via the pattern grammar, a notion of semistructured data schema proposed in the paper.

The main advantages of our approach over others is summarized as follows: first, the ORA-SS data model expresses the semantics of semistructured data more adequately and clearly than any other existing data modeling notation; second, we provide an automatic methodology to validate both the semistructured data schema and instance by utilizing the Alloy System, which has become increasingly crucial with the growing usage of semistructured data.

The remainder of the paper is organized as follows. Section 2 gives a brief overview of the ORA-SS data modeling notation and the Alloy language. Section 3 defines a formal semantics of the ORA-SS data modeling language in Alloy. Section 4 presents a case study that demonstrates the validation of semistructured data using the Alloy Analyzer. Section 5 outlines the conclusions and the directions of future work.

2 Background

2.1 ORA-SS data model overview

The ORA-SS data model is comprised of three basic concepts: object classes, relationship types and attributes. An ORA-SS schema diagram represents constraints at the schema level that any corresponding ORA-SS instance diagram (semistructured data) should follow. For example, in the schema diagram we can represent the constraint that there is a binary many-to-many relationship type between two object classes '`class`' and '`student`'. In the instance diagram we can represent the constraint that there is a relationship between object instance '`class`' with attribute value '`cname:CS105`' and object instance '`student`' with attribute value '`ID:1002`'. An example of these can be found in Figure 2.

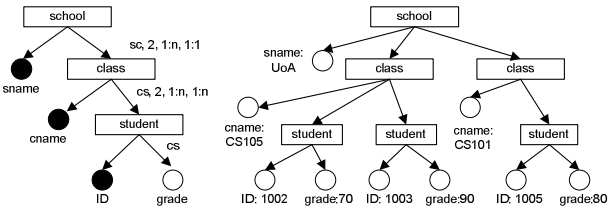

Figure 2. Examples of ORA-SS schema and instance diagrams.

Figure 2 shows an ORA-SS schema diagram and its instance diagram. The diagram on the left represents an ORA-SS schema model. A labelled rectangle represents an object class, and a labelled circle with name represents an attribute. Note that the filled circle in the diagram represents the key attribute of an object class. The relationship between an object and an attribute is denoted by a directed edge, which

means an object has certain attributes, e.g., a 'school' has an attribute called 'sname'. The relationship type among objects is denoted by a directed labelled edge. The label represents a tuple '(name, n, p, c)', where name denotes the name of the relationship, 'n' is an integer indicating the degree of the relationship (e.g. n = 2 indicates binary, n = 3 indicates ternary, etc.), 'p' is the participation constraint of the parent object class in the relationship type, and 'c' is the participation constraint of the child object class in the relationship type. The participation constraint is defined in the form of 'min:max', which represents the lower and upper limit of the participation instances. For example, we can see in the schema diagram that the relationship type between the object classes 'school' and 'class' is give as 'sc, 2, 1:n, 1:1', which indicates that the relationship type is named 'sc' and it is a binary relationship with a '1:n' constraint on 'school' (a school can have 1 or more classes) and a '1:1' constraint on the 'class' (a class belongs to 1 and only 1 school). The label on the edge between an object class and an attribute indicates that the attribute is a relationship type attribute. For example, the label on the edge between 'student' and 'grade' in the schema diagram indicates that 'grade' is an attribute of the relationship type 'cs'.

The diagram on the right represents an ORA-SS data instance, where labelled rectangles represent object instances, labelled circles with names and values represent attribute instances, and the directed edges represent relationship instances among object instances and attribute instances. From the diagram we can see that the instance diagram represents one possible set of instances corresponding to the data model defined by the schema diagram. A full description of the ORA-SS data modeling language can be found in [9, 15].

2.2 Alloy overview

Alloy is a structural modeling language based on first-order logic, suitable for expressing complex structural relationships and constraints. The syntax of Alloy is similar to the standard mathematical syntax of first order logic. The essential constructs of Alloy are as follows [12]:

- A signature ('*sig*' in Alloy syntax) denotes a set of entity objects. It introduces a basic type and a collection of relations (called fields) along with the types of the fields and constraints. A signature may inherit fields and constraints from another signature.
- A fact ('*fact*' in Alloy syntax) is a constraint on relations and objects that always holds. It is a formula that takes no arguments and does not need to be invoked explicitly.
- A predicate ('*pred*' in Alloy syntax) is a template for a parameterized constraint. It can be applied elsewhere by instantiating the parameters and evaluates to either true or false.
- A function ('*fun*' in Alloy syntax) is a template for a parameterized expression. It can be applied elsewhere by instantiating the parameters and evaluates to a value.
- An assertion ('*assert*' in Alloy syntax) is a constraint that is intended to follow from the facts of a model. It is a formula whose correctness needs to be checked, assuming the facts in the model.

Alloy is equipped with the Alloy Analyzer. It translates Alloy specifications into propositional formulas and generates a set of finite scope instances that satisfy the properties expressed in the specifications by exploiting the SAT solvers [20]. Alloy Analyzer provides two kinds of automatic analysis: simulation in which the consistency of a fact or predicate is demonstrated by generating a snapshot of the model; and checking, in which a consequence of the specification is tested by attempting to generate a counterexample for an assertion. Alloy and its analyzer has been successfully applied in various research case studies [8, 10, 17].

3 Formal semantics of ORA-SS data model

In this section, we present an Alloy semantics for the ORA-SS data modeling language. We first define some basic concepts for the notation, followed by other features of the ORA-SS language. Due to the space limit, only part of the semantics is presented in the paper. For a detailed view on the Alloy ORA-SS semantics, please refer to our recent technical report [18].

3.1. Basic concepts

First, we define three basic signatures, Object, Relationship and Attribute, for the schema level description of semistructured data. They represent the concept of object class, relationship type and attribute in the ORA-SS schema diagram respectively.

3.1.1 Object class

The signature of Object is defined as follows:

```
abstract sig Object {
 instances: set OInstance,
 parent: lone Object,
 ancestor: set Object,
 related: lone Relationship,
 has: set Attribute,
```

```
 directhas: set Attribute,
 key: lone Attribute,
 ref: lone Object,
 refed: set Object
}
```

The keyword `abstract` indicates that `Object` will be refined further by other signatures that extend it. The declaration gives all fields for an Object class. For example, in each Object class, it contains a set of `instances` and an optional `parent` object class. The `ancestor` relates to a set of object classes which are above it in the hierarchy. The `related` field defines a possible relationship type. An object class has a set of attributes, which involves the field `has` and the field `directhas`, where `has` is the combination of its own attributes (`directhas`) and its referenced attributes. The `key` defines that each object class can have one identifier only. The last two fields defines the reference relationship in the ORA-SS data model, i.e., each object class can have one referenced object class `ref`, whereas an object class can be referenced by several other object classes `refed`.

Based on the field definitions in the `Object` signature, we can state some basic facts about those fields as follows.

```
{
 all x:Object | x !in (x.^@parent + x.@ref
     + x.@refed + x.@related.@contain)
 ancestor = parent + parent.@ancestor
     + refed + refed.@ancestor
 one related => this.@parent=related.@end
 has = directhas + ref.@has
 one key => O_CandidateA (this, key)
            && key in has
 one ref => one ref.@key && key = ref.@key
 one ref =>this in ref.@refed
 some refed => this = refed.@ref
 one ref => all x:instances |
     one y:ref.@instances |
         x->y in ORASSModel.R_OiOi
 one ref => all x:ref.@instances |
     one y:instances |
         y->x in ORASSModel.R_OiOi
}
```

These facts express constraints on the fields. For example, the second fact states that an `ancestor` should include the `parent` and its own ancestors as well as the ancestors of its references. The fourth fact defines that the attributes of each object class are the combination of its own attributes and its reference's attributes. The fifth fact states that `key` must be one of the candidate key attributes of the object class.

3.1.2 Relationship type

A relationship type can be binary or n-ary (where $n > 2$). The signature of `Relationship` is defined as follows:

```
abstract sig Relationship{
 instances: set RInstance,
 contain: some Object,
 start : one Object,
 end: one Object
}
```

A relationship type can also have a set of `instances`, and it must contain some object classes that participate in the relationship. We can define some basic facts about those fields as follows.

```
{
 start & end in contain
 start != end
 #contain=2 => start= end.@parent
   && no end.@related
 contain in end.*@parent
 #contain>2 => one end.@related
   && contain-end = end.@related.@contain
 instances.@contain in contain.@instances
}
```

The first two facts state that the intersection of `start` object class and the `end` object class must be a subset of the object classes of the relationship type, and the `start` object class of the relationship type cannot be the same as the `end` one. The third fact defines that the `start` object class of the relationship is the parent of the `end` object class if the relationship is a binary one. The next two facts define that the relationship type is a consecutive object class chain that starts from the object class higher in the hierarchy to those lower in the hierarchy. The last fact states the constraint between relationship type and its instances.

3.1.3 Attribute

An attribute may be single or composite, and must belong to either an object class or a relationship type. The signature of `Attribute` is defined as follows:

```
abstract sig Attribute {
 instances: set AInstance,
 related: lone Relationship,
 contain: set Attribute
}
```

An attribute can have a set of valid values as its `instances`, and may be `related` to a relationship type. A composite attribute also can contain other attributes via the field `contain`.

3.1.4 Instances

After giving three basic signatures for the description at the schema level, we define signatures for the instance level description of semistructured data. Having treated `Object`, `Relationship` and `Attribute` as three signatures, all of them will have a set of corresponding instances, i.e., `OInstance`, `RInstance` and `AInstance`. For example, `OInstance` represents the object instances of an object class, and its Alloy representation is defined as follows:

```
abstract sig OInstance {
  parent: set OInstance,
  ancestor: set OInstance,
  has: set AInstance,
  directhas: set AInstance,
  related: set RInstance,
  ref: lone OInstance,
  refed: set OInstance
}
```

We can see that the `OInstance` has similar fields corresponding to the object class signature. However, the difference is that the types of those fields are defined in terms of various instances, e.g., the `parent` field of an object instance is a set of `OInstance` type. Thus the instance signature represents the relationships among instances in an ORA-SS data model. A set of facts that relate to the above instance fields can be defined as follows.

```
{
  no parent && no refed=> no ancestor
  has = directhas + ref.@has
  one ref => this in ref.@refed
  some refed => this = refed.@ref
  this->ref in ORASSModel.R_OiOi
  refed->this in ORASSModel.R_OiOi
  parent->this in ORASSModel.R_OiOi
  this -> has in ORASSModel.R_OiAi
  related->this in ORASSModel.R_RiOi
}
```

The first four facts define the set of constraints similar to that of the object class signature. For example, the first fact states that an object instance will have no ancestor if it has no parent instance and it is not referenced by other object instances. The last five facts of `OInstance` define some default relationships in a `ORASSModel`, which will be explained further in the next subsection. Similarly, the signatures for relationship instance (`RInstance`) and attribute instance (`AInstance`) can be defined [18].

3.2 ORA-SS diagram

The ORA-SS diagram includes all the instance relationships involved in a semistructured data model. The signature of `ORASSModel` is defined as follows.

```
one sig ORASSModel {
  R_OiOi: set OInstance -> OInstance,
  R_OiAi: set OInstance -> AInstance,
  R_AiAi: set AInstance -> AInstance,
  R_RiOi: set RInstance -> OInstance,
  R_RiAi: set RInstance -> AInstance
}
```

The keyword `one` indicates that `ORASSModel` is a concrete type and each ORA-SS schema should have one and only one `ORASSModel` element. The five fields of `ORASSModel` represent five types of relationships among the object instances, relationship instances and attribute instances in an ORA-SS data model. The `R_OiOi` denotes the relationship between an object instance and another object instance. The `R_OiAi` represents an object instance to an attribute instance relationship, where the `R_AiAi` denotes the relationship between two attribute instances. The `R_RiOi` represents the relationship between a relationship type instance to an object instance, and the `R_RiAi` denotes the relationship between a relationship type instance to an attribute instance. As we mentioned earlier, an ORA-SS diagram is defined in terms of these five types of relationship among the different instances. When creating an instance signature, corresponding relationships are established automatically in the `ORASSModel`. For example, in the previous subsection 3.1.4, the last five facts of the `OInstance` signature defines its relationships in `R_OiOi`, `R_OiAi` and `R_RiOi`.

3.3 ORA-SS language constructs

Having presented the ORA-SS basic elements in Alloy, we are ready to define various language constructs of the ORA-SS data modeling language. These ORA-SS language constructs are defined in terms of the parameterized constraints on the five types of relationships in the `ORASSModel`.

3.3.1 Candidate key attribute

The candidate key attribute of an object class refers to an attribute where each of its instance value uniquely identify an instance of the parent object class that holds the attribute. It can be defined in Alloy as follows.

```
pred O_CandidateA(o:Object, a:Attribute) {
  a in o.has
  all x:o.instances | one y:a.instances
            | x->y in ORASSModel.R_OiAi
  all x:a.instances | one y:o.instances
            | y->x in ORASSModel.R_OiAi
  all x:a.instances | x.ancestor = x.parent
       + x.parent.ancestor && no x.decideby
}
```

The above predicate, which includes four constraints, specifies the features that a candidate key attribute must have. The first constraint indicates that a candidate key attribute must be in the field `has` of the object class. The next two constraints specify that the relationships between object instances and candidate key attribute instances are one to one. The last constraint indicates that the object candidate key attribute is related to a single object class rather than a relationship type.

3.3.2 Disjunctive attribute

The disjunctive attribute is an attribute where its instances are selected from a set of attributes. It can be presented in Alloy is as follows.

```
pred DisjunctiveA(p : Attribute,
                  c : set Attribute)
{
  c = p.contain
  p.instances = c.instances
  all x:p.instances |
            x.(ORASSModel.R_AiAi) = x
  all x:p.instances | one y: c.instances
            | x->y in ORASSModel.R_AiAi
  all x:c.instances | one y: p.instances
            | y->x in ORASSModel.R_AiAi
}
```

The above predicate defines five constraints between a disjunctive attribute and its children. The first constraint indicates the children of a disjunctive attribute must be in the set of attribute that the disjunctive attribute contains. The second constraint shows that the instance of a disjunctive attribute is equal to the union of all its children instances. The third constraint states that the disjunctive attribute relationship must be reflexive. The last two constraints indicate that there is always a one to one mapping for each pair of disjunctive attribute instances. Similarly, we can apply the same approach in defining the disjunctive object class and composite attribute constructs in the ORA-SS language.

3.3.3 Participation constraint on object class

In this subsection, we present how the ORA-SS participation constraint of an object class is defined in Alloy. Firstly, the participation constraint on a parent object class with a minimal number is defined as follows:

```
pred OOPMinConstraint (p: Object, c:Object,
                                  min: Int) {
  all x:p.instances |
    #(x <: (p.instances->c.instances &
           ORASSModel.R_OiOi)) >= int min
}
```

Where `p` and `c` represent the parent object class and child object class respectively, and `min` is an integer representing the minimal participation on the parent object class. The constraint specifies that the number of parent instances in the relationship type should be greater than or equal to the minimum. In a similar manner, the participation constraint on parent object with maximum number `OOPMaxConstraint`, the participation constraint on child object with minimum number `OOCMinConstraint` and the participation constraint on child object with maximum number `OOCMaxConstraint` can be defined accordingly.

So far, we have presented some of the Alloy semantics for the ORA-SS language due to the space limit. By using the same approach, a complete set of ORA-SS language constructs can be defined accordingly. A detailed ORA-SS Alloy semantics can be found in [18].

3.4 Presenting the ORA-SS diagram in Alloy

After defining an Alloy formal semantics for the ORA-SS data modeling language, any ORA-SS diagram can be easily represented in Alloy. We encode our Alloy semantics of ORA-SS into a module named '`ORASS`', so that users can import this module when constructing their own ORA-SS schema and instance data models. For example, the simple schema diagram example in Figure 2 of Section 2.1 can be presented in Alloy as follows.

```
open ORASS
sig school, class, student
                      extends Object {}
sig sname, cname, ID, grade
                    extends Attribute {}
sig sc, cs extends Relationship {}
fact {
 one school && one class && one student
 one sname && one cname
 one ID && one grade
 one sc && one cs
 noParent (school)
 class.parent = school
 student.parent = class
 OOPMinConstraint (school, class, Int 1)
 OOCMinConstraint (school, class, Int 1)
 OOCMaxConstraint (school, class, Int 1)
 OOPMinConstraint (class, student, Int 1)
 OOCMinConstraint (class, student, Int 1)
 school.directhas = sname
 class.directhas = cname
 student.directhas = ID + grade
 school.key = sname
 class.key = cname
 student.key = ID
 sc::defineR(school, class, school+class)
```

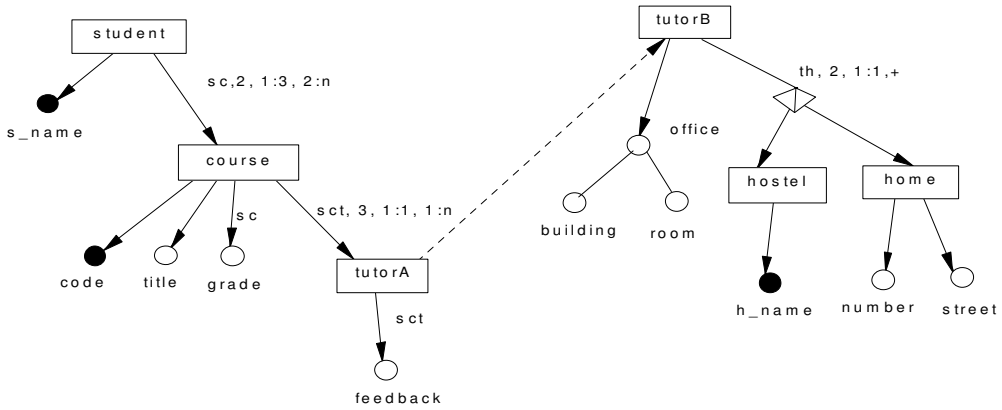

Figure 3. An ORA-SS schema diagram.

```
  cs::defineR(class, student, class+student)
  R_OptionA(cs, grade)
  noRelated (school + class + student)
  noRef (school + class + student)
  noRefed (school + class + student)
  singleAttribute (sname+ cname+ ID+ grade)
}
```

The `open` command imports the `ORASS` module that contains the ORA-SS Alloy semantic definitions. The signatures of `school`, `class` and `student` have been created by extending the object class signature. Similarly, corresponding attribute and relationship type definitions are created. Finally, the fact section specifies the semantic links among the object classes, attributes and relationship types in the diagram, e.g., the `school` is a parent of `class`, the participation constraint from the `class` to `school` is 1 to 1 and so on. After transforming an ORA-SS schema diagram into an Alloy-based formal model, we can then perform automated validation using the Alloy Analyzer. In the next section, we will demonstrate the consistency checking processes through a more complicated case study.

4 Validating ORA-SS data models in Alloy

With the assistance of the Alloy Analyzer, we can perform automated semistructured data validation at both the schema level and instance level. The following is a case study for demonstrating the verification process.

4.1 Schema validation

Figure 3 shows an ORA-SS schema diagram that contains most of the language constructs in the ORA-SS data modeling language. In the diagram, object class `student` is related to object class`course` via the binary relationship type `sc`, and there is a ternary relationship type `sct` between `student`, `course` and `tutorA`. Note that `grade` is an attribute of the binary relationship type `sc` and `feedback` is an attribute of the ternary relationship type `sct`. The object class `tutorB`, which has a composite attribute `office` and a disjunctive relationship type with object class `hostel` and object class `home`, is referenced by the reference object class `tutorA`. We present this ORA-SS schema diagram in the Alloy model '`example_schema`', so that it can be used for instance validation later.

```
module example_schema
open ORASS
sig student, course, tutorA, tutorB, hostel,
                      home extends Object {}
sig s_name, code, title, grade, feedback,
  office, building, room, hours, h_name,
  number, street extends Attribute {}
sig sc, sct extends Relationship {}
fact {
 ...
 student = course.parent
 course = tutorA.parent
 ...
 O_OptionA (tutorB, office)
 CompositedA (office, building+room)
 DisjunctiveO (tutorB, hostel+home)
}
```

In this example, we define schema object classes such as `student`, `course`, etc., which extend the `Object` signature from the `ORASS` module. Similarly, we define the set of attributes and the relationship types used in the schema diagram. We state the facts that describe the relationships in the diagram, such as `student` is the parent of `course`, `course` is a parent of `tutor` and so on. We also define the ORA-SS constructs used in the diagram such as `office` is a composite attribute of `building` and `room`, etc.

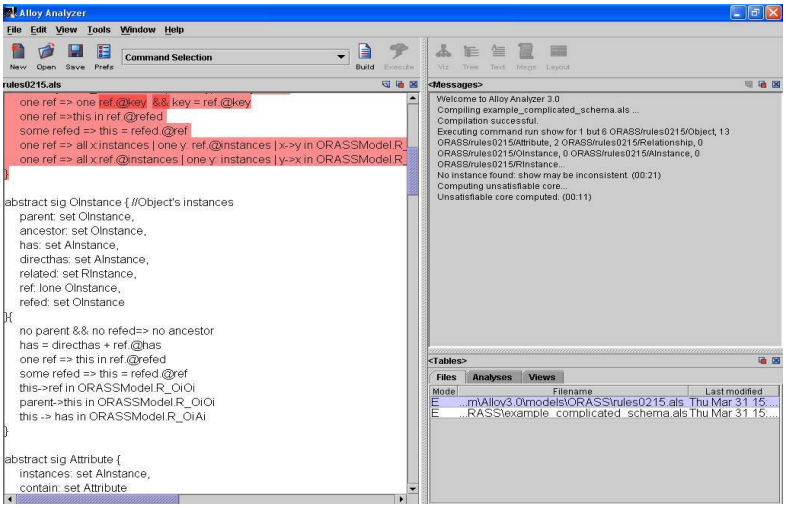

Figure 4. Identifying the source of inconsistency in an invalid schema.

```
pred show () {}
run show for 1 but 6 Object, 13 Attribute,
  2 Relationship, 0 OInstance, 0 AInstance,
  0 RInstance
```

After representing the ORA-SS schema diagram in Alloy, we create a predicate called `show` and perform a `run` command with the number signatures in the model to automatically validate this schema. When executed, Alloy Analyzer will try to create a solution that satisfies all the constraints defined in the model. Alloy outputs '*No solution found, `show` may be inconsistent*', which means the Alloy Analyzer cannot generate a running snapshot of all the defined signatures that satisfy the above model. Thus we may conclude that this ORA-SS schema model is inconsistent because there are conflict constraints in the model. After careful review of the ORA-SS schema diagram in Figure 3, we find that the inconsistency is due to the referenced object class `tutorB` not having a key identifier.

Tracing the source of inconsistency could be frustrating without any tool support. The developers of Alloy Analyzer have added the functionality '*unsatisfied core*' in their latest version, where this functionality can provide some assistance for tracing the cause of an invalid model. For example, Figure 4 shows the result after compute the '*unsatisfied core*' for the above inconsistent schema example. In the code panel on the left, Alloy Analyzer highlights (in red) the facts that caused the inconsistency. After examining these clauses, we find that the inconsistency is due to the fact that every referenced object class must have a key attribute whereas `tutorB` does not, which violates the ORA-SS language semantics defined in the `ORASS` module. This gives the same result as from our manual inspection.

To make this schema valid, we add a new key identifier named `t_name` for the `tutorB` object class in the schema diagram and run the model again. This time Alloy Analyzer outputs '*Instance found*', which means that it has found an snapshot of the signatures that satisfies all the constraints. Based on the generated solution, we can conclude that this ORA-SS schema model is consistent. In Alloy Analyzer, there are three ways to display a solution: graphical view using the visualization menu, a tree structure, or in textual form.

4.2 Instance validation

Besides the schema level validation, we can also validate whether ORA-SS data instances conform with their defined schema model. For example, we can check the validity of the following XML data file against the schema in Figure 3 (with the additional correction of the key attribute `t_name`).

```
<?xml version="1.0" encoding="utf-8" ?>
<uni>
 <student s_name="Tim">
  <course code="c101" title="database"
                          grade="80">
   <tutor t_name="Mike" feedback="good" />
  </course>
  <course code="c102" title="java" grade="90">
   <tutor t_name="Joe" feedback="verygood" />
  </course>
 </student>
 <student s_name="Ben"/>
 <student s_name="Robert">
  <course code="c102" title="java" grade="90">
   <tutor t_name="Joe" feedback="verygood" />
```

```
  </course>
  <course code="c103" title="c" grade="80">
   <tutor t_name="Joe"/>
  </course>
 </student>
 <tutor t_name="Mike" office="Bt100A"
                buliding="B" room="t100A" >
  <hostel h_name="H1"/>
 </tutor>
 <tutor t_name="Joe" office="Bt110A"
                buliding="B" room="t110A" >
  <home number="10C" street="mount" />
 </tutor>
</uni>
```

We first transform this XML representation of the schema instances into its Alloy representation as follows.

```
module example_instance
open example_schema
sig student1, student2, student3, course1,
     ..., home1 extends OInstance{}
sig Tim, Ben, Robert, C101,
     ..., mount extends AInstance {}
...
fact {
 one student1 && one student2 ...
 //assign instances
 student.instances = student1+student2
                     + student3
 course.instances = course1+course2+course3
 tutorA.instances = tutorA1+tutorA2
 ...
 //R_OiAi relationship
 student1->Tim + student2->Ben + ...
   + course2->C102 + course3->C103 + ...
   + home1->n_10C + home1->mount
  = ORASSModel.R_OiAi
 //R_OiOi relationship
 student1->course1 + student1->course2
   + student3->course1 + ...
   + course1->tutorA1 + ...
   + tutorA2->tutorB2 + ...
   + tutorB1->hostel1 + tutorB2->home1
  = ORASSModel.R_OiOi
 //R_AiAi relationship
 Bt100A->B + Bt100A->t100A + Bt110A->B
  + Bt110A->t110A = ORASSModel.R_AiAi
 ...
}

pred show1 (){}
 run show1 for 1 but 6 Object, 13 Attribute,
  2 Relationship, 12 OInstance, 25 AInstance,
  13 RInstance
```

In the above instance representation, after creating the corresponding instances and defining the relationships among the instances, we ask Alloy Analyzer to generate a running example of the system. Alloy Analyzer outputs '*No instance found*' which means this may be an inconsistent set of instances. After reviewing the data instances, we find that it is inconsistent because every student must study at least one course based on the participation constraints defined in the schema definition, whereas the student '`Ben`' has none.

Similarly, the following incorrect instances cannot pass the validation from the Alloy Analyzer.

```
//R_OiAi relationship
 student1->Tim+student2->Ben+student3->Ben
  +course1->C101+course2->C102+course3->C103
  +course1->database+course2->java+course3->c
  +course1->g80+course2->g90+course1->g70
  +course3->g80+tutorA1->Mike+tutorA2->Joe
  +tutorB1->Mike+tutorB2->Joe+tutorA1->good
  +tutorA2->verygood+tutorA1->neutral+
  tutorB1->Bt100A+ tutorB2->Bt110A+
  hostel1->H1+home1->n_10C+home1->mount
 = ORASSModel.R_OiAi
  ...
```

Alloy Analyzer returns the result that it is inconsistent after running. This time it is inconsistent because `s_name` is the key identifier of `student` and there are two students named '`Ben`' in the above example.

4.3 Lessons learned

Undertaking the work presented in this paper has helped to deepen the understanding of the ORA-SS data modeling langauge. Some features in the langauge are not well defined, and representing the ORA-SS data model in Alloy has forced us to precisely define those features. Considering how an XML document is represented in Alloy also forced us to think about what constraints are expressible in XML.

The most important finding is that it is possible to represent the semantics of the ORA-SS data model in Alloy, and automatically validate both the schema and instance levels using the Alloy Analyzer. Although we were successful, we observe that there are potential problems with our current model. In hindsight we could design a model which contains less repetition that is easier to understand. For example, the rules for the schema and the instance are currently intertwined. Another concern is the performance of the Alloy Analyzer when analyzing the current model.

Based on these findings, we are currently developing a new 3-layer representation of the ORA-SS data model in Alloy. The first layer contains semantic constraints on the ORA-SS language, e.g., there must be three object classes associated with a ternary relationship. The second layer models the constraints on the ORA-SS schema and the third layer models constraints on the data instance.

5 Conclusion

In this paper, we presented an approach to provide automatic reasoning support for semistructured data design. We first formalized the ORA-SS data modeling notation in the Alloy formal language. We defined a formal semantics for the ORA-SS data model in Alloy, and then transformed both the semistructured data and its schema into their corresponding Alloy-based ORA-SS models. With the assistance of Alloy Analyzer, we demonstrated that the consistency of the semistructured data and its schema model can be validated automatically. We showed that even subtle errors which are hard to find manually can be easily detected using our approach.

In the future, we will revisit our ORA-SS semantic encoding in Alloy and extend our work to validate the properties of algorithms that are applied to semistructured data. We plan to focus our research on the normalization of semistructured data schema. The normal form of the ORA-SS data model for designing semistructured databases has been proposed in [19]. We would like to validate whether the semantics of a normalized schema is the same as its original schema. This can show whether a normalization algorithm changes the semantics of the schema during the transformation. We also plan to develop a visual environment that can transform XML documents to their Alloy-based ORA-SS models automatically, reducing user interactions and guaranteeing there are no errors in the resulting models.

References

[1] C. Anutariya, V. Wuwongse, E. Nantajeewarawat, and K. Akama. Towards a Foundation for XML Document Databases. In *EC-WEB '00: Proceedings of the First International Conference on Electronic Commerce and Web Technologies*, pages 324–333. Springer-Verlag, 2000.

[2] V. Apparao, S. Byrne, M. Champion, S. Isaacs, I. Jacobs, A. L. Hors, G. Nicol, J. Robie, R. Sutor, C. Wilson, and L. Wood. Document Object Model (DOM) Level 1 Specification. `http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/`.

[3] N. Bidoit, S. Cerrito, and V. Thion. A First Step Towards Modeling Semistructured Data in Hybrid Multimodal Logic. *Journal of Applied Non-classical Logic*, 14(4), 2004.

[4] T. Bray, 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/`.

[5] P. Buneman, S. B. Davidson, M. F. Fernandez, and D. Suciu. Adding Structure to Unstructured Data. In *ICDT '97: Proceedings of the 6th International Conference on Database Theory*, pages 336–350. Springer-Verlag, 1997.

[6] 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.

[7] G. Conforti 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*, pages 37–48. Rubettino Editore, 2003.

[8] G. Dennis, R. Seater, D. Rayside, and D. Jackson. Automating commutativity analysis at the design level. In *ISSTA '04: Proceedings of the 2004 ACM SIGSOFT international symposium on Software testing and analysis*, pages 165–174. ACM Press, 2004.

[9] 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, Singapore, 2001.

[10] J. S. Dong, J. Sun, and H. Wang. Checking and Reasoning about Semantic Web through Alloy. In K. Araki, S. Gnesi, and D. Mandrioli, editors, *FME*, volume 2805 of *Lecture Notes in Computer Science*, pages 796–813. Springer, 2003.

[11] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, and M. A. Jeusfeld, editors, *VLDB'97: Proceedings of 23rd International Conference on Very Large Data Bases*, pages 436–445. Morgan Kaufmann, 1997.

[12] D. Jackson. Micromodels of Software: Lightweight Modelling and Analysis with Alloy, 2002. `http://alloy.mit.edu/reference-manual.pdf`.

[13] D. Jackson, I. Schechter, and I. Shlyakhter. Alcoa: the Alloy Constraint Analyzer. In *ICSE'00: Proceedings of 22th International Conference on Software Engineering*, Limerick, Ireland, June 2000.

[14] 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*.

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

[16] J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):54–66, 1997.

[17] M. Vaziri and D. Jackson. Checking Heap-Manipulating Procedures with a Constraint Solver. In *TACAS '03: Proceedings of 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Warsaw, Poland, 2003.

[18] L. Wang, G. Dobbie, and J. Sun. Formalizing and Validating ORA-SS data model with Alloy. Technical Report UoA-SE-2005-2, Software Engineering, The University of Auckland, New Zealand, 2005.

[19] X. Wu, 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*, Kyoto, Japan, 2001.

[20] H. Zhang. SATO: An Efficient Propositional Prover. In *CADE-14: Proceedings of the 14th International Conference on Automated Deduction*, pages 272–275. Springer-Verlag, 1997.