2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering

Verifying Semistructured Data Normalization using SWRL

Yuan Fang Li
School of ITEE
University of Queensland
liyf@itee.uq.edu.au

Jing Sun, Gillian Dobbie, Scott Lee
Dept. of Computer Science
The University of Auckland
{j.sun, gill, scott}@cs.auckland.ac.nz

Hai H. Wang
School of Eng. & Sci.
Aston University
h.wang10@aston.ac.uk

Abstract

Semistructured data has become more and more prominent in the fast growing areas of web information technology. XML has been used as a standard format for semistructured data in representing and exchanging information in various applications. However, the lack of formality and verification support in the design of a good semistructured data model may hinder its development. For example, redundant data in XML must be removed or minimized to avoid inconsistent and inefficient information processing. Normalization algorithms have been developed to overcome these problems by transforming the schema of a semistructured document into a better form. Therefore, it is essential to ensure that a transformed schema model preserves the same information that its original form holds. In this paper, we present an approach to investigate and verify the no-data-loss property of semistructured data normalization. We encode the verification criteria in the Semantic Web Rule Language (SWRL) and make use of its ontology reasoning engine to provide automated support for the checking process. In summary, our approach not only investigates the information preserving aspect of semistructured data normalization, but also provides a scalable and automated solution towards the problem.

Keywords: *Formal Verification, Semistructured Data Modeling, Semantic Web Rule Language, Ontological Reasoning.*

1 Introduction

Semistructured data has become more and more prominent in the fast growing area of web technologies. The eXtensible Markup Language (XML) has been used as a standard data format for semistructured data in representing and exchanging information in and among various systems. Despite the fast development in the area, questions such as how to design a good semistructured data model still remain a challenge. One of the major concerns in designing a good semistructured data model is to minimize the redundant information stored in the data to achieve consistency and efficient access. Normalization algorithms [1, 2, 13] have been introduced for this purpose by transforming the schema of semistructured data into a better form. From an information preservation point of view, it is essential to ensure that the transformed schema does not lose any information that its original form holds. Unfortunately, existing algorithms for normalizing semistructured data lack of such verification. For instance, the previously mentioned normal forms such as `NF-SS` [13] and `XNF` [2] all claimed to reduce the redundancies in the semistructured data, however, none of them had means of verification to check whether the transformed schema still preserves the same information as that of the original. Without such verification, semantic equivalence of normalization/transformation on semistructured data models cannot be guaranteed.

On the other hand, there are mature algorithms that could be used to verify the no-data-loss property of normalization in relational database systems. As a matter of fact, many normalization algorithms [1, 2] of semistructured data were strongly influenced by their relational counterparts. In this paper, we present an approach to verifying the no-data-loss property for the normalization of semistructured data models. We define our data models in the Object Relationship Attribute model for Semi-Structured data (ORA-SS), which is a semantically enriched data modeling language for semistructured data design [12] and has been used in many XML related database applications [11, 12]. In our previous work [10], we explored the synergy between the semantic web reasoning and semistructured data design. We successfully applied the Web Ontology Language (OWL) [7] and its reasoner to model and verify the correctness of an ORA-SS data model and its XML instances. In this paper, we further advance the approach with a no-data-loss checking facility for the ORA-SS schema normalization. We define the algorithm in OWL and the Semantic Web Rule Language (SWRL) [6], and make use of the Jess rule engine [3] to provide automated reasoning support for the verification process. Figure 1 illustrates an overall pic-

978-0-7695-3757-3/09 $25.00 © 2009 IEEE
DOI 10.1109/TASE.2009.11

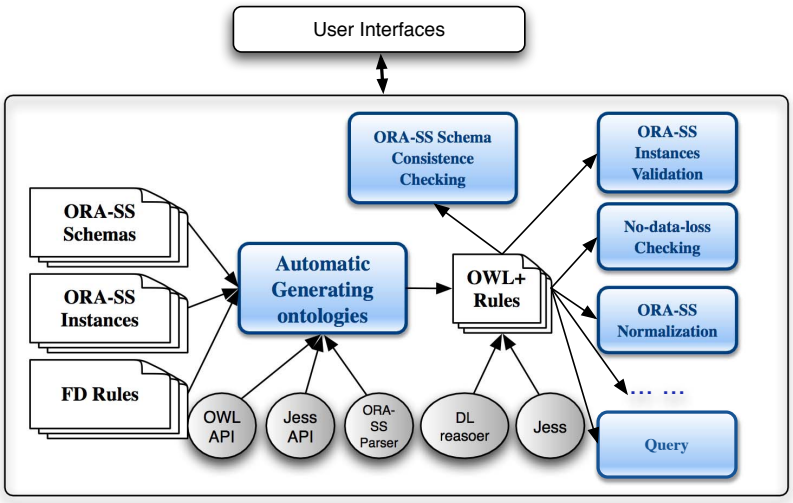

Figure 1. Overall approach to the verification of ORA-SS Schema and its normalization.

ture of our approach towards the verification of the ORA-SS data models and their normalization.

The rest of the paper is organized as follows. Section 2 presents background information on the ORA-SS data modeling language, normalization, the Semantic Web languages OWL & SWRL and the Jess rule engine. Section 3 is devoted to presenting the OWL & SWRL modeling of a no-data-loss checking algorithm for the ORA-SS schema transformation. In Section 4, we demonstrate the automated verification process using the Jess rule engine with a running example of the '`Staff-Project`' ORA-SS schema model. Section 5 concludes the paper and discusses future works.

2 Background

2.1 ORA-SS and normalization

The Object Relationship Attribute model for Semistructured data (ORA-SS) is a semantically enriched data modeling language for semistructured data design [12]. It consists of four basic concepts, i.e., object classes, relationship types, attributes and references.

- An object class represents an entity type in a data model. 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 as a labeled edge by a tuple (*name*, *n*, *p*, *c*), where the *name* denotes the name of relationship type, integer *n* indicates degree of relationship type, *p* and *c* represents participation constraints of parent and child object classes in relationship type.
- Attributes represent properties and are denoted by labeled circles. An attribute can be a key attribute that has a unique value, which is represented as a filled circle. An attribute can be a property of an object class or a property of a relationship type.
- An object class can reference another object class to model recursive and symmetric relationships. This can reduce redundancy especially in many-to-many relationships. References are represented by labeled, dashed edges.

When designing the structure of a semistructured data representation, an ORA-SS schema diagram specifies the relationships, participation and cardinality constraints among the instances of the object classes in a semistructured data model [12]. For example, Figure 2 presents an ORA-SS schema diagram of a simple '`Staff-Project`' data model and its corresponding XML instance. It captures the information of staff members and the projects they are involved. The object class `Employee` has six attributes, i.e., staff ID number (`SSN`), employee name (`EName`), project ID number (`PNumber`), project name (`PName`), project location (`PLocation`), and the number of hours allocated to the project (`Hours`). The XML instance in Figure 2 illustrates an example of the actual data represented.

However, if we closely examine the XML instance, we can easily observe that redundant information exists in the

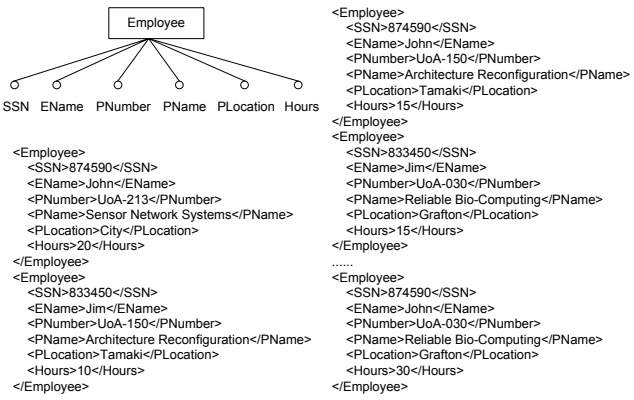

Figure 2. A simple ORA-SS schema of a Staff-Project data model and its XML instance.

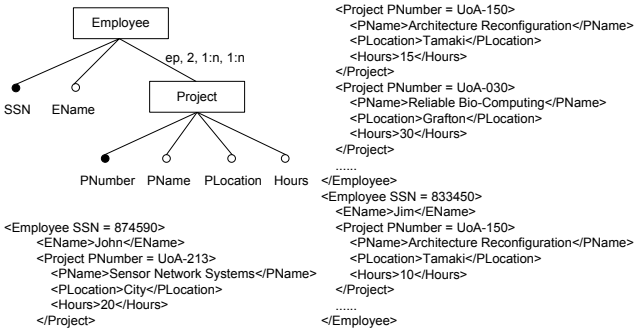

Figure 3. A normalized ORA-SS schema of the Staff-Project data model and its XML instance.

above example, e.g., for each record that represents a staff and a project, the employee's information is repeated in association with the project information. Such redundancy not only wastes the storage space, but also causes possible inconsistencies during insertion and deletion. Therefore, normalization algorithms are applied to prevent this by transforming the original ORA-SS schema into a better form.

Normalization is a process that analyzes and restructures the schema of the semistructured data to minimize redundancies with the help of the functional dependencies of the data model. Let us suppose that the set of functional dependencies for the "`Staff-Project`" ORA-SS schema example in Figure 2 are: (1) staff ID number (`SSN`) determines name of the employee (`EName`), (2) project ID number (`PNumber`) determines project name (`PName`) and project location (`PLocation`), and (3) staff ID number (`SSN`) together with the project ID number (`PNumber`) determines the number of hours allocated (`Hours`) to the project for the staff member. Based on these dependency constraints, we can apply a normalization algorithm to transform the example ORA-SS schema in Fig. 2 into the structure in Figure 3.

In the above ORA-SS schema, each employee in the XML data instance keeps a list of projects that the person involved in, which removes the repeated staff information for each project in the original structure. This reduces redundancy in the file and improves the storage and performance. The example shows that data redundancies can be reduced by transforming the schema of semistructured data into a better form according to a normalization algorithm. However, a normalization process is only effective if the process does not lose or corrupt any information during the schema transformation. Suppose that another normalization of the example ORA-SS schema in Fig. 2 produces the following two object classes $C0$ and $C1$.

$$C0 = [ENAME, PLOCATION]$$
$$C1 = [SSN, PNUMBER, PNAME, PLOCATION, HOURS]$$

Obviously, the data integrity of project number determining its location that is specified by the function dependencies would be lost. Therefore, it is essential for semistructured data normalization to ensure that no-data-loss property of the transformation is preserved.

2.2 Web Ontology Language, Semantic Web Rule Language and Jess Engine

The Web Ontology Language (OWL) [7] provides essential language constructs for describing web resources. Compared to XML, it is one step further towards enabling automated machine processing of online information by marking up web resources with *semantics*. With Description Logics (DL) as the theoretical foundation, OWL ontologies describe web resources, organize them into hierarchies and establish inter-relationships among them.

OWL organizes information into three basic categories, i.e., classes, individuals and properties. Briefly, OWL classes represent abstract domain knowledge. Individuals are concrete entities that belong to specific classes. Properties are binary relations that relate classes, individuals and properties. Table 1 summarizes the "DL syntax" used in the following sections. Interested readers may refer to [7] for full details.

The Semantic Web Rule Language (SWRL) [6] was proposed in 2004 to alleviate the expressivity constraint by adding Horn-style rules to the Semantic Web. The major extensions of SWRL with respect to OWL include Horn-style rules, (universally quantified) variable declarations and a set

Notation	Explanation
$\top/\bot$	Super class/sub class of every class
$N_1 \sqsubseteq N_2$	N_1 is a sub class/property of N_2
$C_1 = C_2$	Class equivalence
$C_1 \sqcap / \sqcup C_2$	Class intersection/union
$\geq 1\ P \sqsubseteq C$	Domain of property P is class C
$\top \sqsubseteq \forall P.C$	Range of P is C
$\top \sqsubseteq \leq 1\ P$	Property P is functional
$\forall / \exists P.C$	allValuesFrom/someValuesFrom restriction, giving the class that for every instance of this class that has instances of property P, the values of the property are all/some members of the class C
$= / \leq / \geq n\ P$	Cardinality restriction, the class each of whose instances mapped by property P forms a set whose cardinality must be exactly/less than/greater than n

Table 1. Summary of OWL syntax in the paper

of pre-determined built-ins for data types. A SWRL rule is a logical implication from an antecedent to a consequent. It is of the form '*antecedent* $\rightarrow$ *consequent*', which means that whenever the conditions specified in the antecedent hold, the conditions specified in the consequent should also hold. Protégé-OWL [5] is a graphical development environment for OWL and SWRL ontologies. It provides the users with the abilities of interacting with OWL and SWRL reasoners such as FaCT++, RACER [4] and Jess [3] to verify ontologies. Because of the high expressivity needs of the no-data-loss property of semistructured data schemas, we will rely on SWRL rules to capture them. As a result, the automated reasoning ability is very important for this approach. We chose the Java Expert System Shell (Jess) [3] as the rule engine to perform the task. It has three components: a rule base, a working memory that contains all the facts and an inference engine that matches rules against the facts. The Protégé-OWL ontology tool bridges the Jess rule engine and translates SWRL rules and the OWL ontology into Jess format, making them available to be reasoned about. The reasoning result (Jess assertions representing new facts) can subsequently be asserted back to the ontology.

3 Modeling no-data-loss property

To verify the no-data-loss property of a semistructured data normalization, we consider each object class and relationship as a relational table with its corresponding key attributes. The algorithm tries to join all the attributes of the object classes and the relationships according to their functional dependencies for the transformed ORA-SS schema. If there exists a set of attributes that produced by the joining of attributes that includes the entire set of attributes of the original ORA-SS schema, we would conclude that the no-data-loss property is satisfied for the transformation. Based on our previous work [10] on validating ORA-SS data models, we further present our modeling and verification of the no-data-loss property using OWL and SWRL [6] as follows.

3.1 Schema modeling

Firstly, we define an OWL class *ROW*. It contains a number of *CELL*s, each of which encapsulates a particular attribute and its associated flag value (a_i or b_{ij}) in the no-data-loss checking algorithm. This is because OWL only supports binary relations (properties). Hence, a number of OWL object-properties (relationships from the domain class to the range class) are defined to link the rows to their cells, as well as from the cells to their respective flags and attributes. The class *FLAG* is an enumerated class with two instances: a and b, which are an abstraction of the a_i and b_{ij} values in the algorithm. The following OWL fragment models these auxiliary classes and object properties, whose domains and ranges are defined.

$$ROW \sqsubseteq \top \qquad FLAG \sqsubseteq \top$$
$$CELL \sqsubseteq \top \qquad FLAG = \{a, b\}$$

$$\geq 1\ hasCell \sqsubseteq ROW \qquad \geq 1\ hasFlag \sqsubseteq FLAG$$
$$\top \sqsubseteq \forall\ hasCell.CELL \qquad \top \sqsubseteq \forall\ hasFlag.FLAG$$

$$hasOneAttribute \sqsubseteq hasAttribute$$
$$\geq 1\ hasOneAttribute \sqsubseteq CELL$$
$$\top \sqsubseteq \forall\ hasOneAttribute.ATTRIBUTE$$

The schema (*ROW*) was constrained to contain at least one cell and all its cells must be instances of the class *CELL*. These constraints are modeled using OWL class restrictions.

$$ROW \sqsubseteq \forall\ hasCell.CELL$$
$$ROW \sqsubseteq \geq 1\ hasCell$$

Similarly, cells also needs to be constrained for the flags and the attributes that they map to.

$$CELL \sqsubseteq \forall\ hasOneAttribute.ATTRIBUTE$$
$$CELL \sqsubseteq = 1\ hasOneAttribute$$

$$CELL \sqsubseteq \forall\ hasFlag.FLAG$$
$$CELL \sqsubseteq = 1\ hasFlag$$

OWL class restrictions are used to state the fact that each cell has exactly one flag value and one attribute. Moreover, the fact that each cell can have only one unique value

for attributes and flags makes the mapping (i.e., the two properties *hasCell* and *hasOneAttribute*) from cells to attributes/cells functional. We also defined *hasOneAttribute* as a sub property of the object property *hasAttribute* in [10].

$$\top \sqsubseteq \leq 1\ hasFlag$$

$$hasOneAttribute \sqsubseteq hasAttribute$$
$$\top \sqsubseteq \leq 1\ hasOneAttribute$$

Note that since the property *hasOneAttribute* applies to *CELL*, the domain of its super property and itself need to be augmented.

$$\top \sqsubseteq \forall\, hasAttribute.(ENTITY \sqcup CELL)$$
$$\top \sqsubseteq \forall\, hasOneAttribute.CELL$$

According to the no-data-loss checking algorithm, a transformation is no-data-loss if one of the rows is shown to contain all the attributes (*a*'s) in the schema. Hence, another OWL datatype property is needed. This boolean-valued property, called $no-data-lossRow$, applies to rows and denotes whether this particular row is no-data-loss or not. Note that the range of $no-data-lossRow$ is *boolean*, a datatype defined in XML schema.

$$\geq 1\ no-data-lossRow \sqsubseteq ROW$$
$$\top \sqsubseteq \forall\, no-data-lossRow.boolean$$

3.2 Functional dependency modeling

Functional dependencies (FDs) for semistructured data can be viewed as functions from a set of attributes to another set of attributes. The FDs are represented using OWL classes together with SWRL rules. As stated in Section 2, the `Staff-Project` example defines three FDs, one of them is that the social security number of an employee determines his/her name: $\{SSN\} \rightarrow ENAME$. In our modeling, the OWL class *FUNCTIONALDEPENCENCY* represents the class of place holders for FDs. Each instance of *FUNCTIONALDEPENCENCY* is mapped to a set of attributes via the property *hasAttribute* (its domain needs to be updated). These attributes define the domain of the FD. Each *FUNCTIONALDEPENCENCY* is mapped to the range attribute of the FD by an object property *determines*. Moreover, each instance of class *FUNCTIONALDEPENCENCY* is constrained to have at least one *determines* fact.

$$FUNCTIONALDEPENCENCY \sqsubseteq \top$$
$$\top \sqsubseteq \forall\, hasAttribute.(ENTITY \sqcup CELL \sqcup FUNCTIONALDEPENCENCY)$$

$$FUNCTIONALDEPENCENCY \sqsubseteq \forall\, hasAttribute.ATTRIBUTE$$
$$FUNCTIONALDEPENCENCY \sqsubseteq \geq 1\ hasAttribute$$

$$\top \sqsubseteq \forall\, determines.ATTRIBUTE$$
$$\geq 1\ determines \sqsubseteq FUNCTIONALDEPENCENCY$$

$$FUNCTIONALDEPENCENCY \sqsubseteq \forall\, determines.ATTRIBUTE$$
$$FUNCTIONALDEPENCENCY \sqsubseteq \geq 1\ determines$$

For instance, the above FD '$\{SSN\} \rightarrow ENAME$' is modeled using the following OWL fragment.

$$SSN \in ATTRIBUTE \qquad ENAME \in ATTRIBUTE$$
$$FD1 \in FUNCTIONALDEPENCENCY$$
$$\langle FD1, SSN \rangle \in hasAttributes$$
$$\langle FD1, ENAME \rangle \in determines$$

In addition, the transitivity property of the FDs states that if a member of the range of a FD is also the domain of another FD, then the range of the second FD also depends on the domain of the first. This cannot be modeled directly in OWL, but can be modeled using a SWRL rule as follows.

$$FUNCTIONALDEPENCENCY(?xs) \wedge FUNCTIONALDEPENCENCY(?ys) \wedge determines(?xs, ?y) \wedge hasOneAttribute(?ys, ?y) \wedge determines(?ys, ?z) \rightarrow determines(?xs, ?z)$$

3.3 No-data-loss checking modeling

After a careful examination of the no-data-loss checking algorithm, we realized that the expressivity of OWL language does not suffice to model it. Hence, we resort to its Horn-style rules extension, the SWRL [6] language for this task. In the following, we present the detailed modeling of this algorithm based on the definitions from the previous section. The no-data-loss checking algorithm is procedural, which involves repeatedly checking against all FDs and setting the flags of some of the cells in a main loop until no further changes can be made. This algorithm can be naturally translated into SWRL by developing a number of rules, each correspond to one of the FDs of the ORA-SS schema. In general, the algorithm can be modeled in the following steps.

1. As stated in the previous section, each *cell* is mapped to a flag value of either *a* or *b*, an abstraction of the a_i and b_{ij} values in the original algorithm. Hence, the first step is to set all those flag values to *b*.

2. For each row_i and each attribute $Attr_j$, if row_i contains (via the intermediate OWL class *CELL*) $Attr_j$, set the flag value of that cell *a*.

3. For each FD $X \rightarrow Y$ and each attribute y in the range Y, we construct a SWRL rule in such a way that for

each row that has the symbol a for all the attributes in X, if any of the rows have the symbol a for the y attribute, set the flag corresponding to the y cell to a. Otherwise, the symbol b is set for the y cell. The rules are repeatedly applied until no change is possible.

4. Another SWRL rule is developed to check whether there exists a row whose entire cells have the a flag. If so, denote the row no-data-loss. If any of the rows is denoted no-data-loss, then the schema is no-data-loss.

As we can see, the above steps closely correspond to the no-data-loss checking algorithm. In the following section, we will present the modeling of the `Staff-Project` example in OWL/SWRL and show how the verification can be performed by using Protégé and the Jess reasoning engine to check the no-data-loss property of the normalization.

4 Verifying ORA-SS schema normalization

4.1 The Staff-Project example

In the example of the `Staff-Project` schema model in Section 2, there are six attributes: SSN, ENAME, PNUMBER, PNAME, PLOCATION and HOURS. It contains three functional dependencies.

$$\{SSN\} \rightarrow ENAME \quad (1)$$
$$\{PNUMBER\} \rightarrow \{PNAME, PLOCATION\} \quad (2)$$
$$\{SSN, PNUMBER\} \rightarrow HOURS \quad (3)$$

The following OWL definitions model the above attributes using OWL classes.

$SSN \in ATTRIBUTE$
$ENAME \in ATTRIBUTE$
$PNUMBER \in ATTRIBUTE$
$PNAME \in ATTRIBUTE$
$PLOCATION \in ATTRIBUTE$
$HOURS \in ATTRIBUTE$

The modeling of the FDs requires the instantiation of the FUNCTIONALDEPENDENCY class, where each FD correspond to one instance of the class.

$FD1 \in FUNCTIONALDEPENCENCY$
$FD2 \in FUNCTIONALDEPENCENCY$
$FD3 \in FUNCTIONALDEPENCENCY$

Given the above definitions, we establish the connection between FDs to their respective domain and range attributes via the object properties defined in Sec. 3. For example, the following fragment show the definition of FD 3 '$\{SSN, PNUMBER\} \rightarrow HOURS$'. The first two restrictions state that the domain of the FD is *SSN* and *PNUMBER* and the third restriction states that the *HOURS* is determined by the two attributes in this FD. Other FDs can be similarly modeled.

$\langle FD3, SSN\rangle \in hasAttribute$
$\langle FD3, PNUMBER\rangle \in hasAttribute$
$\langle FD3, HOURS\rangle \in determines$

Following the modeling of FDs, the decomposition can be modeled using rows and cells. Each row corresponds to one object class or relationship type in the ORA-SS schema diagram. Each row has n cells, where n is the total number of attributes in the schema. For instance, in the `Staff-Project` example, n would be 6 and it consists of three decompositions.

$R0 = [SSN, ENAME]$
$R1 = [PNUMBER, PNAME, PLOCATION]$
$R2 = [SSN, PNUMBER, HOURS]$

The decomposition $R0$, denoted by $row0$ in the OWL model, can be modeled as follows.

$row0 \in ROW \qquad \langle row0, false\rangle \in no-data-lossRow$

$cell\_0\_0 \in CELL \qquad \langle row0, cell\_0\_0\rangle \in hasCell$
...
$cell\_0\_5 \in CELL \qquad \langle row0, cell\_0\_5\rangle \in hasCell$

$\langle cell\_0\_0, SSN\rangle \in hasOneAttribute$
$\langle cell\_0\_0, a\rangle \in hasFlag$
...
$\langle cell\_0\_5, HOURS\rangle \in hasOneAttribute$
$\langle cell\_0\_5, b\rangle \in hasFlag$

In the above definition, $row0$ is marked as '*false*' (not no-data-loss) initially. Six cells $cell\_0\_0$ to $cell\_0\_5$ are defined as instances of *CELL*. They are mapped to $row0$ using the property *hasCell*. Moreover, each of these cells is mapped to the attribute it represents with its flag value also set to 'b'. According to the algorithm, a cell's flag value is set to 'a' when its corresponding attribute is present in the relationship (row). Hence, the flag values of the first two cells ($cell\_0\_0$ and $cell\_0\_1$) in $row0$ are set to 'a'.

The above definitions also enables us to model the no-data-loss checking algorithm using SWRL rules. As stated in the previous section, each FD corresponds to one or more SWRL rules, depending on the number of attributes in the range of the FD. For example, FD 2 '$PNUMBER \rightarrow \{PNAME, PLOCATION\}$' has two attributes in its range,

thus two SWRL rules are defined. Rule *FD*2_0_*rule* corresponds to FD 2 for the attribute *PNAME*. The code fragment below shows the rule *FD*2_0_*rule*[1].

$$FUNCTIONALDEPENCENCY(FD2) \wedge \quad (1)$$
$$hasAttribute(FD2, PNUMBER) \wedge \quad (2)$$
$$determines(FD2, PNAME) \wedge \quad (3)$$
$$ROW(?x) \wedge \quad (4)$$
$$hasCell(?x, ?y) \wedge hasOneAttribute(?y, PNUMBER) \wedge \quad (5)$$
$$hasFlag(?y, a) \wedge \quad (6)$$
$$hasCell(?x, ?z) \wedge hasOneAttribute(?z, PNAME) \wedge \quad (7)$$
$$hasFlag(?z, a) \wedge \quad (8)$$
$$ROW(?a) \wedge \quad (9)$$
$$hasCell(?a, ?b) \wedge hasOneAttribute(?b, PNUMBER) \wedge \quad (10)$$
$$hasFlag(?b, a) \wedge \quad (11)$$
$$hasCell(?a, ?c) \wedge hasOneAttribute(?c, PNAME) \quad (12)$$
$$\rightarrow \quad (13)$$
$$hasFlag(?c, a) \quad (14)$$

The first 3 lines make sure that the rules are for *FD*2 and that this FD has the correct domain and range attributes. Lines 4 to 8 tries to locate a row that all its flag values for cells corresponding to the attributes in *FD*2, i.e., '*PNUMBER*' and '*PNAME*' are '*a*'s. If such a row is found, due to the no-data-loss algorithm, this row can be used to determine the flag values of the cells in another row that contains the right hand side of the FD. Lines 9 to 12 attempt to locate another row $?a$[2], where the flag values of its cells corresponding to the domain attributes of the FD, i.e., '*PNUMBER*', is '*a*'. If such row $?a$ exists, the rule on line 13 will assert the fact that the cells in row $?a$ that corresponds to the range value, i.e., '*PNAME*' should have '*a*' as its flag value as well, as shown in line 14.

Similarly, a rule *FD*2_1_*rule* corresponds to FD 2 and the attribute *PLOCATION* can be defined. Due to the page limit, we omit its definition here. The complete model of the above example and the base model defining the underlying ORA-SS entities and the checking algorithm can be found at `http://www.itee.uq.edu.au/~liyf/ora-ss/no-data-loss-checking/index.html`. Four such rules are formulated for each FD in the schema diagram. In addition, another SWRL rule is defined to locate a row that has flag value *a* for all its cells. If such a row is found, then we can conclude that the particular normalization (decomposition) is no-data-loss.

[1]For the ease of reference, line numbers are listed.

[2]Note that $?a$ is a SWRL variable and '*a*' is an OWL individual belonging to the class *FLAG*.

4.2 Checking with Jess rule engine

The model defined previously enables us to perform automated reasoning on ORA-SS semistructured data model using the SWRL reasoning engine - Jess [3]. The Jess reasoning engine has been adapted to provide reasoning support for the SWRL language through the SWRLJessTab, a plugin for Protégé-OWL. As stated in Section 2, SWRLJessTab imports all SWRL rules and relevant OWL ontology definitions and performance inference on the imported knowledge base. At the end of the execution, newly inferred facts are asserted back to the OWL ontology. Figure 4 shows the imported SWRL rules and OWL ontology fragments imported by the Jess rule engine after pressing the `OWL+SWRL->Jess` button on the bottom left. All six SWRL rules, together with some OWL class, individual, property definitions and restrictions were imported.

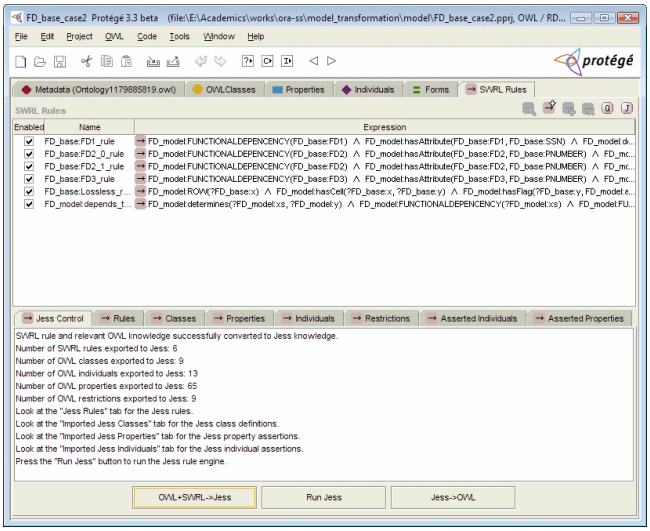

Figure 4. Rule & ontology imported into Jess

By invoking the `Run Jess` function, seven properties (i.e., property assertions) were asserted by the rule engine. They are shown in the 'Asserted Properties' tab in Figure 5.

Figure 5. Execution result of the Jess Engine.

As we can see from the diagram , 6 out of the 7 assertions evaluate their flag values of the cells to '*a*'. However, one assertion, `(assert (FD_model:no-data-lossRow row2 "true"))`, proves that *row*2 is actually satisfied

the no-data-loss property. With this assertion, we can conclude that the normalization (transformation) satisfies the no-data-loss property. This property can be subsequently asserted back to the OWL ontology by pressing the `Jess->OWL` button. Hence, the reasoning result is recorded in the ontology. This last step makes subsequently querying of the no-data-loss property of the model faster by avoiding repeated application of the checking algorithm.

The above demonstrates the ability of the Jess engine in verifying the no-data-loss property. It is also capable of showing whether a transformation has data loss. For example, using the same set of attributes and functional dependencies, if the normalization were defined as $C0$ and $C1$ previously mentioned in section 2.1, it can be proved that the transformation was not no-data-loss. Jess engine shows that by *not* asserting that any of the rows can be evaluated to '*true*'.

5 Conclusion

With the rapid growth of semistructured data usage, the needs for designing good semistructured data models become more and more critical. Normalization is a process that analyzes and transforms the schema of the semistructured data model to minimize redundancies and improve efficiency. It is essential to ensure that the transformed schema does not lose any information that the original form holds. In this paper, we present an automated and scalable approach towards the data preserving aspect of verifying the correctness of semistructured data normalization. We defined the no-data-loss property in the semistructured data context and encoded the checking algorithm using OWL/SWRL, which made use of its reasoning engine to provide automated tool support for the checking process. Note that the checking only needs to be evaluated once, as the Jess engine asserts the proved properties back into the knowledge base for future data query purpose. Furthermore, the approach for verifying semistructured data transformation can be used to prove the correctness of different normalization algorithms developed for semistructured data. Furthermore, the declarative description of the methodology in SWRL also provides a solid stepping stone for the automated verification of the normalization algorithms used in the semistructured database systems. Compared with our recent work in verifying semistructured data design using PVS [9, 8], the translation from ORA-SS into OWL was more direct, as there is a close co-relation between semantic web ontologies and semistructured data models. Most importantly, the verification was far more scalable, since the semantic web reasoners are designed to handle huge sets of data instances. We believe that this is a significant advantage over other approaches.

In the future, we plan to extend our work in defining algorithms to verify the dependency preserving aspects of the normalization for semistructured data, and provide automated and scalable verification support using OWL/SWRL reasoners. We also plan to develop a systematic mapping tool for supporting the translation from the ORA-SS models into their corresponding OWL/SWRL counterparts to further automate the verification process.

References

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

[2] 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.

[3] E. Friedman-Hill. *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA, 2003.

[4] V. Haarslev and R. Möller. *RACER User's Guide and Reference Manual: Version 1.7.6*, Dec. 2002.

[5] A. R. Holger Knublauch, Mark Musen. Editing description logic ontologies with the protégé-owl plugin. In *International Workshop on Description Logics - DL2004*, 2004.

[6] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. `http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/`, May 2004.

[7] I. Horrocks, P. F. Patel-Schneider, and F. van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *J. of Web Semantics*, 1(1):7–26, 2003.

[8] S. U.-J. Lee, G. Dobbie, J. Sun, and L. Groves. Formal Verification of Semistructured Data Models in PVS. *Journal of Universal Computer Science, Special Issue on Logic, Abstract State Machines and Databases*, 15(1):241–272, 2009.

[9] S. U.-J. Lee, J. Sun, G. Dobbie, and L. Groves. Verifying Semistructured Data Normalization using PVS. In *ICECCS '08: Proceedings of the 13th IEEE International Conference on Engineering of Complex Computer Systems*, pages 15–24, Los Alamitos, CA, USA, April 2008. IEEE Computer Society.

[10] Y. F. Li, J. Sun, G. Dobbie, H. H. Wang, and J. Sun. Reasoning About ORA-SS Data Models Using the Semantic Web. *Journal on Data Semantics VII*, 4244:219–241, November 2006.

[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*, 2001.

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

[13] 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)*, Yokohama, Japan, November 2001. Springer-Verlag.