A Pattern For Interrelated Numerical Properties

Jesper Zedlitz[1], Hagen Peters[2], and Norbert Luttenberger[2]

[1] German National Library of Economics (ZBW)
[2] Christian-Albrechts-Universität zu Kiel

Abstract. "A childs year of birth is always greater than the year of birth of its parents." – it is not easily possible to code this simple knowledge into a pure OWL ontology, i.e. without using any additional rule languages. Therefore it is not easy in OWL to detect semantic violations in this kind of statements. The two challenges are putting two orders ("greater" and "parent") into relation and representing integers as individuals allowing a reasoner to infer knowledge about the "greater" relation. In the first part of this contribution we show a pattern for putting two transitive and asymmetric orders into a relation, such that conflicting information results in an inconsistent ontology. In the second part we present a pattern for expressing integers using their binary code. Due to the special construction a reasoner can infer knowledge about the relation between all integers in the ontology. By combining the two patterns we are able to represent the initial statement in an ontology.

1 Introduction

The Web Ontology Language (OWL) makes a deliberate distinction between object properties of objects and data properties of objects. Data properties are designed to take a certain value from a range that is defined by the associated data type. However, the OWL specification does not go beyond sheer value assignment—no other operations are foreseen for the values of data properties. During the design phase of OWL 2, a number of authors therefore brought forward "wish lists" for different kind of operations on the values of data properties:

– Pan and Horrocks [1] propose the idea to enable calculations with data property values.
– In Use Case #10 of the W3C Working Draft [2] the value—not just the presence—of a data property is intended to be used for classification of individuals into classes.
– Grau et al.[3] list four kinds of operations to be provided for data property values. Among these is the requirement that it should be possible to express relations between values of data properties on different objects.

Although OWL's current version OWL 2 brought a number of enhancements to data type handling, the situation basically remained the same: the cited W3C Working Draft states that none of these wishes has been accepted for OWL 2.

Instead—in order to separate logical reasoning and handling of data values—the decision was taken to concentrate all data value handling inside the Semantic Web Rule Language (SWRL)[1] and its "built-ins". Integrated processing of OWL ontologies and SWRL rule sets accordingly requires software systems that comprise both a reasoner and an oracle for these built-ins. However, it obviously depends on the oracle's implementation how SWRL rules are evaluated (open vs. closed-world processing, data types). Furthermore the semantics of SWRL built-ins is outside the OWL ontology semantics.

In this paper, we present Logical Ontology Design Patterns (ODP) [4] for evaluating certain relations between values on different objects and for representing values from the range of integers as individuals in the ontology. Using these patterns semantic violations in data sets can be detected during consistency checks.

This paper is organized as follows: In section 2 we describe the class of problems we want to address and discuss other approaches to the problem. For clarity we split our solution into three separate parts: Section 3 presents an ODP for putting two orders into a relation. Section 4 presents our ODP for representing integers. In section 5 we introduce a third ODP that combines both previous ODP and prove the correctness of our approach. Section 6 concludes and points out open questions. We give examples written in OWL 2 functional-style syntax [5] where appropriate.

2 Problem Description & Approach

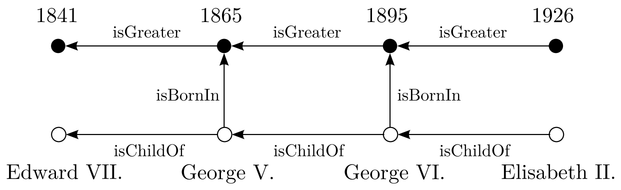

Fig. 1. Example of the problem we address.

As an example for the kind of problems we want to address let us take a look at this situation: A person P_1 has an *isChildOf* relation to another person P_2. Each of the persons has a relation *isBornIn* to its year of birth. This information is coded in an OWL 2 ontology. The ontology shall only be consistent if the year of birth of P_1 is greater than the year of birth of P_2: "A child's year of birth is always greater than the year of birth of its parents." Figure 1 illustrates this example.

There are several ways known to model the above mentioned situation and to express the relation of birth years of parents and their children. One could

[1] http://www.w3.org/Submission/SWRL/

use rule languages like SWRL to express this knowledge. But there are only a few reasoners that (fully) support rules at all and many different rule languages are used. Furthermore, as mentioned in the introduction, at least parts of the semantics of the rule languages is outside of the semantics of OWL. Therefore the evaluation of these parts is rather asking an oracle (i.e. using the specific implementation) than OWL reasoning. For example, the specification of data types in SWRL allows for different implementations in terms of precision of decimals.[2] Hence two different reasoners, both supporting the same parts of OWL and being compliant to the SWRL specification could evaluate the same ontology differently.

For this reason we focus on approaches that only use pure OWL evaluation. There are two main approaches to express relations between natural numbers (in the following "integers" w.r.t. common data type definitions) as in our example: representing integers as literals and representing integers as individuals.

Representing integers as literals The common way to model the above scenario with integers represented as literals is to use data properties for the year of birth and a combination of restrictions on object and data types to detect semantic violations. Listing 1 shows a restriction for the individual named "GeorgeV" requiring his father's birth year to be before 1865.

```
1  Declaration( NamedIndividual( :EdwardVII ) )
2  Declaration( NamedIndividual( :GeorgeV ) )
3  ClassAssertion(
4      ObjectAllValuesFrom( :isChildOf
5          DataAllValuesFrom( :isBornIn
6              DatatypeRestriction( xsd:integer xsd:maxExclusive
7                  "1865"^^xsd:integer )
8          )
9      )
10     :GeorgeV
11 )
12
13 DataPropertyAssertion( :isBornIn :GeorgeV "1865"^^xsd:integer )
14 DataPropertyAssertion( :isBornIn :EdwardVII "1841"^^xsd:integer )
15 ObjectPropertyAssertion( :isChildOf :GeorgeV :EdwardVII )
```

Listing 1. Using data properties and data type restrictions

However, although this approach seems more or less obvious it suffers from two major disadvantages:

1. No **general** statements about birth years of parents and children are made, but for each child the maximum birth year of its parents has to be specified. That requires additional axioms for each individual that belongs to the ordered set.
2. Now there is a restriction on the parent's birth year but there is **no** formal correspondence between this restriction (line 7) and George's birth (line 13). That means, the ontology could also be consistent if one restricts the parent's

[2] *"All minimally conforming processors must support decimal numbers with a minimum of 18 decimal digits"* from http://www.w3.org/TR/xmlschema-2/ section 3.2.3

birth year to a maximum value of 1865 (line 7) while (in line 13) George's birth year is (e.g. by mistake) set to 1800, which is obviously not intended.

In summary, with this approach we are not able to express a general statement about the relation of the years of birth of parents and their children but have to express that knowledge for each of the parent-child relations explicitly (1). Furthermore, this approach still allows for semantic violations (2).

Representing numbers as individuals Other authors propose the use of resources (i.e. OWL individuals) rather than literals for the representation of numbers. [6] shows several advantages of this approach. Most interesting for our problem is the possibility to reason about relations between these number individuals and other individuals of the ontology. However, in this approach the name (IRI) of an individual is used to encode some knowledge about the resource. Since names of individuals are meaningless character sequences in terms of formal reasoning, it is a priori not possible to use the knowledge encoded in the individual's names during the reasoning process.

Another approach for representing an integer n is to specify the predecessor (and/or successor) of n. Using this approach implies, that if a particular integer n is needed in the ontology all $n - 1$ predecessors must be part of the ontology, too. Thus the representation of an integer depends on other individuals that are (or aren't) contained in the ontology. This obviously implies that adding an integer representing individual is a non-trivial task and requires full knowledge of the ontology.

It might also be possible to represent an integer n by using an individual having n properties and adding appopiate cardinality constraining axioms. In both approaches, using predecessors as well as n properties, the number of axioms needed to represent a single integer scales linearly with the value of the integer.

Thus these approaches require large maintenance effort (adding statements about numbers not actually used, changing the definition of previously defined numbers, etc.) and knowledge about already existing integers in the ontology. Furthermore the linear dependency between the value of an integer and the number of axioms needed in the ontology makes these approaches impractical for many scenarios.

Our approach In our approach we want to use individuals to represent integers. Our goal is to find a pattern where

1. the number of axioms needed for representing a single integer depends only logarithmically on the value of that integer (like the usual binary or decimal representation),
2. the representation of an integer is independent of whether or not other integers already exist in the ontology, and
3. it is possible to reason about integers based on their representation.

Once we found that pattern we'll be able to detect "direct" semantic violation of the order of integers, e.g. having two integers 3 and 4 and an the explicit statement like "3 is greater than 4".

However, that is not enough for our initial problem, i.e. detecting semantic violations on birth years of parents and their children. In this problem we want to detect "indirect" semantic violations, e.g. person P_1 is the child of person P_2, **but** P_2 is born before P_1. In other words, we have to detect semantic violations between two orders (the order given by birth years *isGreater* and the order given by the child relation *isChildOf*) that are connected by another property (*isBornIn*).

In the next section we show a pattern putting two orders into relation. Section 4 describes our pattern for integer representation.

3 Comparing Two Orders

The first part of our solution is a pattern for putting two arbitrary orders into relation. To outline that this pattern is not restricted to numerical values we use another example here. In Fig. 2 the individuals depicted with black-filled circles represent (in this case non-numeric) time data. The *isYoungerThan* object property establishes an order on these individuals. Analogously the individuals depicted with white-filled circles (in Fig. 2: tools) were put into an order using the *isSuccessorOf* property. The *isToolOfThe* object property connects individuals of the one with individuals of the other order, i.e. tools with ages.

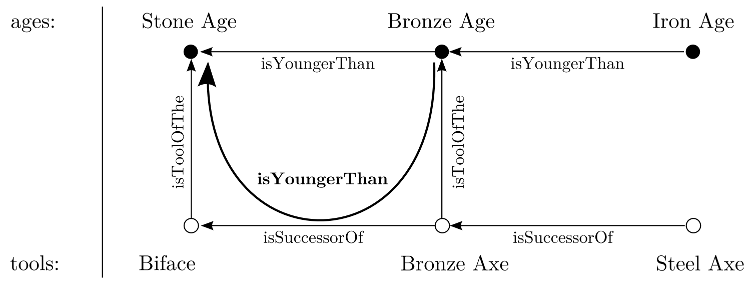

Fig. 2. Comparing two ordered sets of individuals.

The ontology should be inconsistent if two individuals in one order are connected to two individuals with inverse order. The trick is to use one of the order-building properties to infer knowledge about the other one. This can be achieved by using a chain of object properties:

$$isToolOfThe^{-} \circ isSuccessorOf \circ isToolOfThe \rightarrow isYoungerThan$$

To compare not only neighboring individuals in each order it is necessary to declare transitivity for (at least) one of the relations. In our example this would be:

$$isYoungerThan(x,y) \wedge isYoungerThan(y,z) \rightarrow isYoungerThan(x,z)$$

3.1 Asymmetry

Furthermore, if contradicting information should result in inconsistency it is necessary that the following holds:

$$\forall x,y : isYoungerThan(x,y) \implies \neg isYoungerThan(y,x)$$

This could be achieved by marking the $isYoungerThan$ object property asymmetric. Unfortunately this is not possible in OWL 2, because to guarantee decidability[5, sec. 11.2] an object property must not be transitive and asymmetric at the same time.

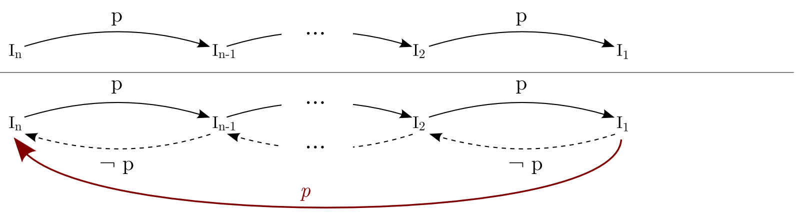

Fig. 3. (Negative) Object Property Assertions.

However, for the known individuals in the ontology it is possible to assure asymmetry by inserting negative object property assertions. To ensure that $\neg p(I_a, I_b)$ holds for arbitrary $b > a$, surprisingly only $n-1$ negative object property assertions are necessary for n ordered individuals: The upper half of Fig. 3 shows the situation where object properties (depicted by solid arrows) are used to represent an order on individuals $I_1 \ldots I_n$. To ensure asymmetry on $I_1 \ldots I_n$ it is sufficient to only insert negative object property assertions between neighboring individuals (depicted by dashed arrows in the lower half of Fig. 3):

Assume the assertion $p(I_1, I_n)$ is part of the ontology sketched above (Fig. 3). Then the following inferences can be made:

$$p(I_1, I_n) \xrightarrow{p(I_n, I_{n-1})} p(I_1, I_{n-1}) \xrightarrow{p(I_{n-1}, I_{n-2})} \ldots \xrightarrow{p(I_3, I_2)} p(I_1, I_2)$$

This is a contradiction to $\neg p(I_1, I_2)$. $\square$

Thus for every x, y with $isYoungerThan(x,y)$ holds $\neg isYoungerThan(y,x)$, stated either explicitly or implicitly.

3.2 Summary I

We have now defined a pattern that enables us to put two orders defined on two sets of individuals—not necessarily integers—into relation. If one of the orders contains individuals representing integers and the relation is a natural ordering on integers this technique solves part of our initial problem. It is just necessary to list the needed integers in the ontology instead of all integers in between. If n integers are used, $2(n-1)$ object properties are required to model the order in the ontology. A semantic violation in the specified ordering makes the ontology inconsistent.

The main disadvantage of this pattern is the fact that an individual has no formal relation to the value it is intended to represent (except for the label which is not part of the knowledge included in reasoning process). If one needs to insert an individual representing a specific value (lets say the "copper age" in Fig. 2) this is not possible without additional knowledge about the already existing individuals (e.g. the individual labeled "bronze age" is younger than "copper age"). Without additional knowledge it is only possible to insert an individual that is known to represent a new minimal or maximal value.

4 Integer Representation

In the second part of our solution we use a pattern that constructs classes containing information about the relation of integers in the ontology. We represent an integer in the binary numeral system with a predefined bit-length. Figure 4 shows every four-digit integer as a leaf of a binary tree (the gray shaded areas can be ignored for now). The binary representation of an integer is given by the path from the root to the leaf. The natural "greater than" order is given by the order the leaves appear in a depth-first tree traversal.

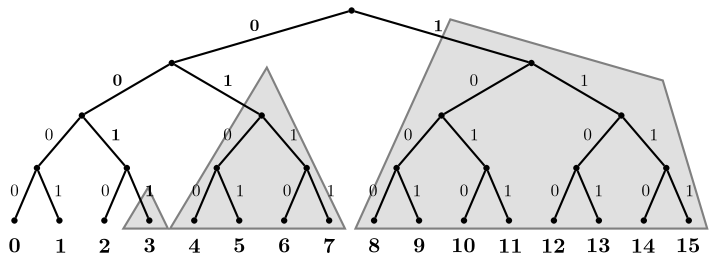

Fig. 4. Binary tree.

Let I_n denote an individual that represents the integer n in the ontology.[3] Let $C_{i.0}$ denote the class of integers with the i^{th} bit being zero. Let $C_{i.1}$ denote the class of integers with the i^{th} bit being one. (The least significant bit is bit zero.)

[3] We expect that there could be more than one individual in the ontology representing a given integer.

We put each required integer into its respective classes, e.g. for the individual "two".

$$I_2 \in C_{3.0}, I_2 \in C_{2.0}, I_2 \in C_{1.1}, \text{and } I_2 \in C_{0.0}.$$

```
ClassAssertion( :3.0 :Two)
ClassAssertion( :2.0 :Two)
ClassAssertion( :1.1 :Two)
ClassAssertion( :0.0 :Two)
```

Listing 2. Representing the 4-bit integer 2

Note that "two" is just an arbitrary label for an individual representing integer 2. It would not make any difference to label the individual e.g. "MyPersonalTwo".

Now we can define a class that contains all individuals that represent the same integer by an intersection of the classes of the binary digits, e.g. for the integer 2:

$$C_{equal.2} = C_{3.0} \cap C_{2.0} \cap C_{1.1} \cap C_{0.0}$$

```
SubClassOf(
     ObjectIntersectionOf(:3.0 :2.0 :1.1 :0.0 )
     :Equal2 )
```

Listing 3. Class containing all individuals representing integer 2

It is easy to see that two individuals representing the same integer n in the ontology belong to the same class $C_{equal.n}$. We note this observation as

Lemma 1 *Let I_a be an individual in the ontology representing the integer a and I_b be an individual representing the integer b. Let $a = b$. Let the ontology contain axioms following the construction rules mentioned above. Then $C_{equal.a}(I_b)$ is also entailed by the ontology.*

The class $C_{greater.n}$ (which contains all integers greater than n) are unions of sub-trees of our binary tree (see Fig. 4): According to the construction rules of the tree it holds for every sub-tree that all leafs that are descendants of the right child node ("1") are greater than every leaf that is a descendant of the left child node ("0").

Thus whenever the path representing an integer n follows a 0-edge all leafs that can be reached via the corresponding 1-edge must be included into $C_{greater.n}$. The set of leafs that can be reached via the 1-edge can be easily written as an intersection of binary classes.

As an example consider the construction of $C_{greater.2}$. The elements of the union can clearly be seen in Fig 4.

$$C_{greater.2} \subseteq C_{3.1} \cup (C_{3.0} \cap C_{2.1}) \cup (C_{3.0} \cap C_{2.0} \cap C_{1.1} \cap C_{0.1})$$

```
SubClassOf(
     ObjectIntersectionOf( :3.1 )
     :GreaterThan2 )
SubClassOf(
     ObjectIntersectionOf( :3.0 :2.1 )
```

```
    :GreaterThan2 )
SubClassOf(
    ObjectIntersectionOf( :3.0 :2.0 :1.1 :0.1 )
    :GreaterThan2 )
```

Listing 4. Class containing all individuals representing integers greater than 2

For every integer n used in our ontology we have to declare that any integer that is equal n must not be greater than n:

$$C_{equal.n} \cap C_{greater.n} = \emptyset$$

```
DisjointClasses( :GreaterThan3 :Equal3 )
```

Listing 5. Definition of strictly greater

Following these construction rules knowledge about the relation of integer representing individuals can be inferred: An individual representing an integer greater than n belongs to the class $C_{greater.n}$. We note this observation as

Lemma 2 *Let I_a and I_b be two individuals in the ontology representing integers a and b (with $b > a$). Let the ontology contain axioms following the construction rules mentioned above. Then $C_{greater.a}(I_b)$ is also entailed by the ontology.*

Summary II

In contrast to the restriction of the first pattern of our solution the second pattern does not require any knowledge about the existing integers in the ontology when adding an individual representing an integer. It is only necessary to follow the named construction rules. Individuals representing integers are automatically put into relation with each other. There is only a linear relation in the number of assertions needed to represent an integer and the number of bits used for the binary representation. The setting of a bit length specifies a maximal number. However, using a defined bit length is very common for computer systems.

The pattern introduced above establishes only relations between individuals and classes not between individuals and object properties which are needed for our initial problem. We address this "gluing problem" in the next section.

5 Putting It Together

Now we can use the information from the classes as constructed in section 4 to establish an order on the integer individuals. The main problem is to create a link between the *greater* object property and classes. Our solution has been inspired by [7]. Formally written we need:

$$\begin{aligned} C_{greater.n} = \{x | \exists y : greater(x, y) \wedge C_{equal.n}(y)\} \\ \cup \{x | \exists y : greater(x, y) \wedge C_{greater.n}(y)\} \end{aligned}$$

It is not possible to state this linking of object properties and classes directly in OWL 2. However, by carefully using the *EquivalentClasses* class axiom and the class expressions *ObjectUnionOf* and *ObjectSomeValuesFrom* it is possible to make an equivalent statement. In OWL 2 functional syntax this axiom has to be added:

```
EquivalentClasses(
    :GreaterThanY
    ObjectUnionOf(
        ObjectSomeValuesFrom( :greater :EqualY )
        ObjectSomeValuesFrom( :greater :GreaterThanY ) ) )
```

Listing 6. Connection between the *greater* object property and classes

If this axiom and the class definitions shown above are included in the knowledge base, the existence of a *greater* property between two individuals classifies the individuals into the according "GreaterThan" classes. This knowledge about the *greater* relation can be used in combination with the first part of our approach. For the generic example sketched in Fig. 5 the following axioms would be part of the corresponding ontology:

```
Declaration( Class( :Number ) )
Declaration( ObjectProperty( :s ) )

Declaration( ObjectProperty( :r ) )
ObjectPropertyRange( :r :Number )

SubObjectPropertyOf( ObjectPropertyChain( ObjectInverseOf(:r) :s :r )
                     :greater )
```

Listing 7. Infer *greater* property

All integer individuals which are connected by the above mentioned property chain are put into a greater relation. Based on this relation the integer individuals are classified into the "GreaterThan" classes. Thereby individuals and object properties are "glued".

As shown in section 4 the integer individuals are already classified into "GreaterThan" classes based on their construction. Thus a semantic violation regarding the order of individuals or the connection to the individuals representing integers leads to contradicting classifications. In combination with the disjoint classes axioms this leads to an inconsistency in the ontology. In the next section we show this formally.

5.1 Proof

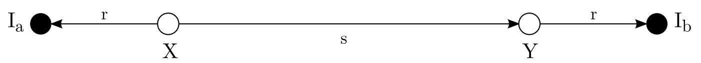

Fig. 5. Sketch of the ontology used in the proof. The "order" is given by object property s, the individuals are X and Y and the integers are I_a and I_b. X and Y are connected to the integers via property r.

To show the correctness of our approach we consider an ontology constructed following the rules above and sketched in Fig 5. It consists of two individuals X and Y that are connected via an object property $s(X, Y)$. The two individuals I_a and I_b represent two integers a and b. X and Y are connected with these integers representing individuals via an object property r.

```
DisjointClasses( :GreaterThanA :EqualA )
EquivalentClasses(
    :GreaterThanA
    ObjectUnionOf(
        ObjectSomeValuesFrom( :greater :EqualA ) )
        ObjectSomeValuesFrom( :greater :GreaterThanA ) ) )

DisjointClasses( :GreaterThanB :EqualB )
EquivalentClasses(
    :GreaterThanB
    ObjectUnionOf(
        ObjectSomeValuesFrom( :greater :EqualB ) )
        ObjectSomeValuesFrom( :greater :GreaterThanB ) ) )
```

Listing 8. Application of listings 5 and 7 for the example of Fig. 5.

Listing 8 can be written formally as:

$$C_{greater.a} \cap C_{equal.a} = \emptyset \tag{1}$$

$$C_{greater.a} = \{x | \exists y : greater(x, y) \wedge C_{equal.a}(y)\} \tag{2}$$

$$\cup \{x | \exists y : greater(x, y) \wedge C_{greater.a}(y)\} \tag{3}$$

$$C_{greater.b} \cap C_{equal.b} = \emptyset \tag{4}$$

$$C_{greater.b} = \{x | \exists y : greater(x, y) \wedge C_{equal.b}(y)\} \tag{5}$$

$$\cup \{x | \exists y : greater(x, y) \wedge C_{greater.b}(y)\} \tag{6}$$

Using the property chain (described in section 3) axiom (cf. listing 7) we can infer:

$$r(X, I_a)^{-} \wedge s(X, Y) \wedge r(Y, I_b) \Rightarrow greater(I_a, I_b) \tag{7}$$

By construction $C_{equal.a}(I_a)$ and $C_{equal.b}(I_b)$ always hold. According to the ontology depicted in Fig. 5 in every case $greater(I_a, I_b)$ is inferred (formula 7). We have to distinguish three cases:

1. $a > b$: $C_{greater.b}(I_a)$ (lemma 1)

$$greater(I_a, I_b) \wedge C_{equal.b}(I_b) \overset{(5)}{\Longrightarrow} C_{greater.b}(I_a)$$

2. $a = b$: $C_{equal.a}(I_b)$ (lemma 1)

$$greater(I_a, I_b) \wedge C_{equal.a}(I_b) \overset{(2)}{\Longrightarrow} C_{greater.a}(I_a)$$

   This is a contradiction to (1) with $C_{equal.a}(I_a)$.

3. $a < b$: $C_{greater.a}(I_b)$ (lemma 2)

$$greater(I_a, I_b) \wedge C_{greater.a}(I_b) \overset{(3)}{\Longrightarrow} C_{greater.a}(I_a)$$

   This is a contradiction to (1) with $C_{equal.a}(I_a)$.

Thus, the semantic violations in cases (2) and (3) result in an inconsistent ontology.

6 Conclusion

In this contribution we present a pattern for putting two orders into relation and a second pattern for representing integers as individuals of an ontology. Following the simple construction rules allows for inference of relations between these individuals. The combination of both patterns can be used to detect contradicting information regarding the two orders. If the ontology contains contradicting information the ontology will become inconsistent. The correctness of our combined pattern is shown in section 5.1.

Our patterns might be beneficial for verification purposes. When testing the performance of the consistency check using test ontologies it turns out that our pattern is not adequate for representing huge amounts of different integers. However, on the one hand in many use cases the time complexity is not worse than a solution that uses only data properties and on the other hand many use-cases do not require many different integers, like the one sketched in Fig. 2.

The authors are aware of the fact that it is possible—if you do not follow the construction rules for integer individuals and relations between them—to create an ontology that is consistent although the represented information contains a semantic violation—for example if statements in listing 2 do not match statements in listings 3 and 4. A solution would be to infer the relations between integer-representing individuals directly from the representation of these integers, but there are some indications that this is not possible. However, it is an open problem to prove this.

Another interesting question would be to investigate if it is possible to model the relation between more than two integers using OWL 2. That could be used to express and check relations that include arithmetic operations. If it should be possible, variants of the initial statement could also be expressed: "A child's year of birth is always at least 10 years greater than the year of birth of its parents."

References

1. Pan, J.Z., Horrocks, I.: Web Ontology Reasoning with Datatype Groups. In: Proc. of the 2nd International Semantic Web Conference. (2003) 47–63
2. Golbreich, C., Wallace, E., Patel-Schneider, P.: OWL 2 Web Ontology Language: New Features and Rationale. W3C working draft (2009)
3. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel-Schneider, P., Sattler, U.: OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web **6**(4) (2008) 309–322
4. Gangemi, A., Presutti, V.: Ontology design patterns. Handbook on Ontologies (2009) 221–243
5. Motik et al. (eds.): OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation (2009)
6. Champin, P.: Representing data as resources in RDF and OWL. Proc. of the 1st Workshop on Emerging Research Opportunities for Web Data Management (2007)
7. Tsarkov, D., Sattler, U., Stevens, M., Stevens, R.: A solution for the Man-Man problem in the Family History Knowledge Base. In: Proc. of the 5th International Workshop on OWL: Experiences and Directions. (2009) 23–24