Incremental Reasoning

Project Outline

Y.Lu and P.J.McBrien

Saturday 8th February 2014

This project aims to refactor the SQOWL system to enable this system to support incremental reasoning when data is not only inserted but also deleted. Incremental Reasoning means when an ontology is updated, only the reasoning task related to this update is performed and the whole semantic reasoning process does not need to be processed again.

SQOWL is a Java-based reasoning system which performs semantic reasoning inside an RDBMS. It divides semantic reasoning into classification of TBox processed by a tableaux reasoner (e.g. Pellet) and type inference of ABox performed by using SQL triggers. OWL classes and properties are mapped into relational tables with one and two columns, respectively. Next, SQOWL uses a tableaux reasoner (e.g. Pellet) to perform classification of TBox and based on the classified TBox, it creates SQL triggers to implement each of the TBox rules. Then, the RDBMS becomes an auto reasoning system, which means when ABox assertions are inserted into the database, the triggers related to these insertions will be invoked to infer other possible insertions into the database.

For example, we have a TBox rule Man $\sqsubseteq$ Person, which denotes a SubClassOf relationship between classes Man and Person. In the database, the two classes are represented as two relational tables namely, Man and Person. The TBox rule is represented by an SQL after trigger (written in Transact-SQL):

```
CREATE TRIGGER Man_isa_Person
ON Man
AFTER INSERT
AS BEGIN
   INSERT INTO Person(id)
   SELECT id FROM inserted
   EXCEPT SELECT id FROM Person
END
```

Now, we consider an ABox rule Man (John), which defines John is a Man. If we insert this assertion into the above database by the following SQL statement:

```
INSERT INTO Man(id) VALUES ('John')
```

The after trigger will be automatically invoked and insert the value John into the table Person.

Obviously, based on the above brief introduction of SQOWL, this system supports incremental reasoning in terms of insertion; however, retraction (i.e. incremental reasoning in terms of deleting) is still not handled yet. Two main rules should be followed in terms of deletion:

- Inferred information should not be deleted unless the information which deduces this inferred data is removed.
- If one piece of information is successfully deleted, the data inferred by this information should be deleted as well if no other information deduce this data.

One possible proposal for solving retraction could be:

1. Enhance the database schema to add another column into each relational table to denote whether a tuple is implicitly inferred or explicitly inserted.

2. Design and implement triggers before a deletion to check whether this deletion is allowed to be deleted.

3. Design and implement triggers after a deletion to further delete possible inferred information.

Based on the above example, the proposal can be implemented by first adding a column called exp into tables Man and Person, and we can use the value 1 of this column to denote a tuple is explicitly inserted (the value 0 means the opposite). Therefore, when John is inserted into the database, the exp of John is 1 in the table Man and is 0 in the table Person, as follows:

Man

id	exp
John	1

Person

id	exp
John	0

Then, we can create triggers before deleting to restrict that only tuples whose exp values are 1, Hence, on the table Man we can have an INSTEAD OF TRIGGER as follows:

```
CREATE TRIGGER insteadof_delete_Man
INSTEAD OF DELELTE
AS BEGIN
  IF EXISTS (SELECT * FROM deleted WHERE exp = 1)
  BEGIN
     RAISERROR('The inserted value contains inferred information', 10, 1)
     ROLLBACK TRANSACTION
     RETURN
  END
END
```

Moreover, in this particular example, since the class Man is a Person, before deleting tuples from the table Person, a trigger should be created to check the deleted tuples are not in the table Man. Thus, the trigger before deleting from the table Person can be:

```
CREATE TRIGGER insteadof_delete_Person
INSTEAD OF DELELTE
AS BEGIN
  IF EXISTS (SELECT * FROM deleted WHERE exp = 1)
  BEGIN
     RAISERROR('The deleted value contains inferred information', 10, 1)
     ROLLBACK TRANSACTION
     RETURN
  END

  IF EXISTS (SELECT * FROM deleted JOIN Man ON deleted.id = Man.id)
  BEGIN
     RAISERROR('The deleted value contains individuals from its subclasses', 10, 1)
     ROLLBACK TRANSACTION
     RETURN
  END
END
```

However, this proposal is not totally able to solve equivalence relationships in an ontology, and one possible solution could be use SQL views to denote the equivalence relationships rather than storing repeated data.

Finally:

- The above proposal is only a possible idea, which might not the best solution for solving the issue of retraction. Therefore, you not need to exactly follow this proposal and can investigate your own ideas.

- You are expected to have several basic knowledge about OWL ontologies, Description Logics, database triggers and Java.