CREATING CUSTOM RULES FOR JENA

HENRIETTE HARMSE

In this post I will show you

1. how to add your own custom rules to Jena,
2. how to use rule primitives, and
3. I will mention some things you may want to keep in mind when using rules.

1. Add and Active Rules

We assume we start with the following simple triples

```
@PREFIX : <http://org.jena.rules.tutorial#> .

:Peet :takesCourse :ComputerScience .
:Ruth :teachesCourse :ComputerScience .
```

for which we add the following rule to a `student.rules` file:

```
@PREFIX : <http://org.jena.rules.tutorial#> .

[hasStudentRule:
  (?student :takesCourse ?course) (?lecturer :teachesCourse ?course)
    -> (?lecturer :hasStudent ?student)]
```

The `hasStudentRule` says that if a student takes a course and that course is presented by some lecturer, then the student is a student of that lecturer. The `(?student :takesCourse ?course) (?lecturer :teachesCourse ?course)` part of the rule is referred to as the **premise** or **body** of the rule and the `(?lecturer :hasStudent ?student)` part as the **conclusion** or **head** of the rule.

To activate the rules in Jena we have to create a model that is capable of doing inferences. In the context of Jena inferencing means that it will re-evaluate the rules against the data, which will cause additional statements to be potentially added to the model. This can be achieved by calling methods that will cause Jena to re-evaluate the model, i.e. calls like `InfModel.rebind()` and `InfModel.validate()`.

```
Path path = Paths.get(".").toAbsolutePath().normalize();

// Load RDF data
String data = path.toFile().getAbsolutePath() +
  "/src/main/resources/data.ttl";
Model model = ModelFactory.createDefaultModel();
model.read(data);

// Load rules
String rules = path.toFile().getAbsolutePath() +
  "/src/main/resources/student.rules";
Reasoner reasoner = new GenericRuleReasoner(Rule.rulesFromURL(rules));
```

Date: 21st April 2018.

```
InfModel infModel = ModelFactory.createInfModel(reasoner, model);
infModel.rebind();
```

When the `hasStudentRule` is activated a new statement will be added to the Jena model:

```
:Ruth :hasStudent :Peet .
```

2. Using Rule Primitives

Jena supports a number of builtin rule primitives that are intuitive to understand, i.e.

```
[pensionerRule:
  (?person :hasAge ?age) greaterThan(?age, 60)
    -> (?person a :Pensioner)]
```

which states that when `?person` has an age greater than 60, `?person` is considered to be a pensioner. Assuming we have the data

```
:Peet :hasAge 90 .
```

the following triple will be added to the Jena model:

```
:Peet a :Pensioner .
```

3. Things to Keep in Mind

There are two main things I think one needs to keep in mind with Jena rules:

1. The purpose of rules are to manipulate the triples in the Jena model. For the most part it adds triples to the model, but it can also remove triples from the model if primitives like `remove` and `drop` are used.
2. Adding and removing of triples can be achieved through SPARQL queries which can perform better or worse than rules. It is therefore best to check both approaches for your given use case.

4. Conclusion

In this post I gave a brief introduction in how to use custom rules and builtin rule primitives in Jena. This code is available at github.

References