Forward Chaining Reasoning Tool for Rya

Rya Working Group, 6/29/2016

OWL Reasoning

OWL (the Web Ontology Language) facilitates rich ontology definition over RDF data. The semantics of the OWL provide a set of inference rules for complex logical reasoning.

Reasoning strategies can typically be classified as either:

- Forward chaining: Proactively apply inference rules to the data
- Backward chaining: Apply inference rules to each query

For example, Rya can perform some backward chaining inference at query time by rewriting queries. This is useful because it only does reasoning related to the query itself, but is limited in scope to avoid overly complex queries.

Motivation

We would also like to perform **offline consistency checking**: Determine whether a potentially large dataset containing rich OWL schema yields any contradictions under the inference semantics.

This problem lends itself to forward chaining:

- Validating the consistency of the entire datastore requires reasoning over all the data
- We are less concerned by the usual challenge (for forward chaining reasoners) of how to maintain stored triples

Goal: A MapReduce tool to perform distributed consistency checking over a large Rya datastore.

Reasoning Using Rules

Full OWL is computationally intractable for large datasets. *OWL profiles* (named *EL*, *QL*, and *RL*) are subsets of OWL 2 designed to trade some expressivity for efficiency.

OWL RL is intended for scalable rule-based reasoning. RL can be expressed and implemented as a set of if/then rules,[1] which take in triples and yield triples or inconsistencies.

Examples

If ?x rdf:type ?c and ?c rdfs:subClassOf ?d, then ?x rdf:type ?d .

If ?x ?p ?x and ?p rdf:type owl:IrreflexiveProperty, then the knowledgebase is inconsistent .

We process a more restricted subset of OWL RL.

[1] https://www.w3.org/TR/owl2-profiles/#Reasoning_in_OWL_2_RL_and_RDF_Graphs_using_Rules

Strategy

Every rule's if-clause is a conjunction of triples. In order to apply a reasoning rule, we must detect all the triples in the if-clause.

If we distinguish between *schema triples* (the relationships between classes and properties) and *instance triples*, then most (but not all) RL rules have the property that **there exists a variable in the rule's if-clause such that every instance triple in the if-clause has that variable as its subject and/or object**.

Therefore, we can distribute reasoning among multiple computation nodes if we ensure that:

- Every node has access to the schema.
- Instance triples are partitioned based on their subject and object.

Example

If every node has schema knowledge:

- :subOrganizationOf rdf:type owl:TransitiveProperty .
- :subOrganizationOf rdf:domain :Organization .
- :Organization owl:disjointWith :Person .

And one node has all triples involving Y:

- :X :subOrganizationOf :Y .
- :Y :subOrganizationOf :Z .
- :Y rdf:type :Person .

Then that node can draw conclusions for its local neighborhood:

- :X :subOrganizationOf :Z . [via the semantics of transitivity]
- :Y rdf:type :Organization . [via the semantics of domain]
- :Y's types are inconsistent. [via disjoint classes]

Supported OWL Constructs

OWL RL vocabulary constructs that can be handled this way include:

Property types

owl:AsymmetricProperty
owl:IrreflexiveProperty
owl:SymmetricProperty
owl:TransitiveProperty

Property restrictions

owl:allValuesFrom
owl:hasValue
owl:someValuesFrom

Relationships among classes and properties

owl:complementOf
owl:disjointWith
owl:equivalentClass
owl:equivalentProperty
owl:inverseOf
owl:propertyDisjointWith
rdfs:domain
rdfs:range
rdfs:subClassOf
rdfs:subPropertyOf

MapReduce Reasoning

Mapper:

- Input: A triple from Accumulo.
- Processing: Based on the schema, decide whether the triple could be relevant to reasoning around its subject and/or object.
- Output: $< subject, triple >$ if determined relevant to subject, and/or $< object, triple >$ if relevant to object.

Reducer:

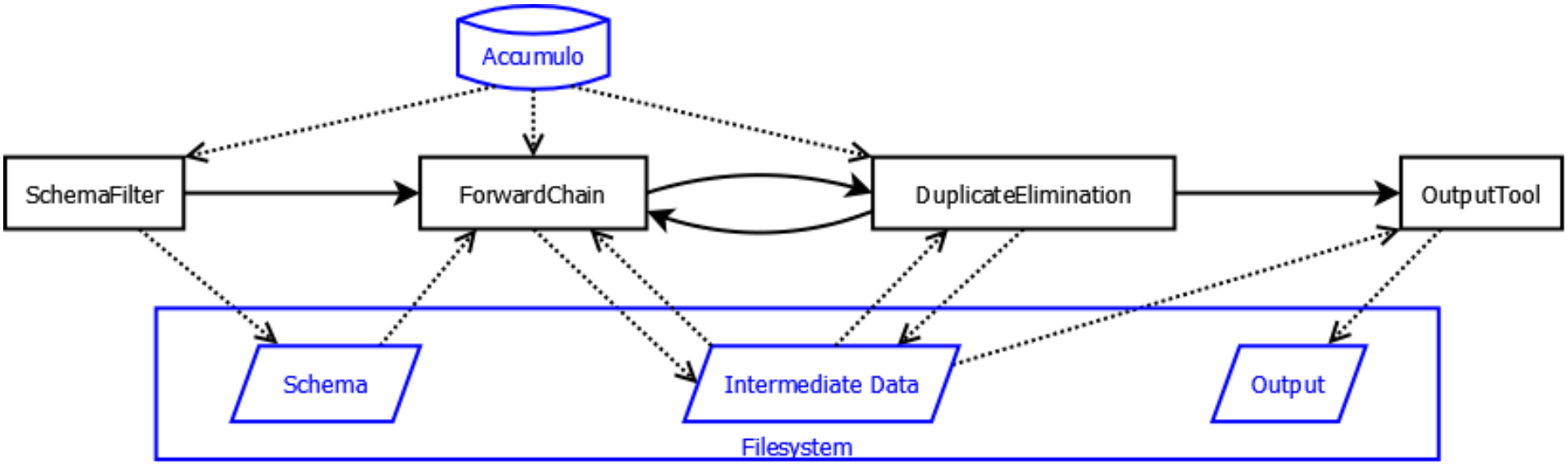
- Input: Resource x and all relevant triples connected to x.
- Processing: Apply RL reasoning rules to x's neighborhood.
- Output: Inferred triples and/or inconsistencies, with the triples that implied them.

Inferred triples can trigger rules for neighboring resources, so we repeat until convergence.

MapReduce Workflow

Implemented as several MapReduce jobs, invoked via **ReasoningDriver** in the **rya.reasoning** project.

1. Collect schema information to distribute to all nodes [SchemaFilter]
2. Repeat until no new information can be derived:
   1. Perform local reasoning [ForwardChain]
   2. Prune duplicate triples and inconsistencies [DuplicateElimination]
3. Collect inferred triples and inconsistencies [OutputTool]



Example Output

Inferred triples

```
<http://example.org/Y> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Organization> .
<http://example.org/X> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://example.org/Organization> .
<http://example.org/X> <http://example.org/subOrganizationOf> <http://example.org/Z> .
```

Inconsistencies

```
Inconsistency:
[owl:disjointWith -- Resource can't belong to two disjoint classes]
   |
   +---<http://example.org/Y>
   |   <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
   |   <http://example.org/Organization>
   |   [rdfs:domain -- Predicate's domain implies subject's type]
   |       |
   |       +---<http://example.org/Y>
   |           <http://example.org/subOrganizationOf>
   |           <http://example.org/Z>
   |           [input]
   |
   +---<http://example.org/Y>
       <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
       <http://example.org/Person>
       [input]
```

Future

- Extend to remaining OWL RL rules:
  - Rules involving *owl:sameAs*
  - Rules involving list constructs, e.g.*owl:unionOf*
  - Rules requiring multi-way joins, e.g.*owl:propertyChainAxiom*
  - Type-checking literals
- Check for unsatisfiable classes: classes that would be inconsistent if they had any members
- Persist results:
  - Store inferred triples in Rya? Should differentiate from original data
  - Update inferences when data changes
  - Combine forward-chaining with backward-chaining for query evaluation
- Explore user-defined rules (SWRL?)