Salmon Run

Swimming upstream on the technology tide, one technology at a time. A collection of articles, tips, and random musings on application development and system design.

Labels

.net acegi actor actorfoundry actorsguild ajax alfresco algorithms annotations ant apache-cxf aspectj atom awk berkeley-db bitfauna bitsets blogger burlap c# caching cascading cassandra cherrypy classification clisp clojure cloud clustering cms collections15 collocation commons-beanutils commons-collections commons-digester commons-httpclient commons-math concept-mapping concurrent conditional-probability content-management cosine-similarity crawler cusp cx_oracle data-management data-mining data-structure

Saturday, June 13, 2009

Ontology Rules with Prolog

Over the years, I've had an on-again, off-again interest in Rules Engines. However, as Martin Fowler points out, it is often more pragmatic to build a custom engine. A custom engine can be as simple as a properties file modelled after an awk script (ie, {pattern => action} pairs). More complex rules, ie multiple pattern matches in a certain sequence leading to a single complex action, can also be modeled by doing a Java variant of the awk strategy, ie {Predicate => Closure}. Where Rules Engines shine, however, is when you need to do rule chaining or when the structure of the rules themselves (rather than their values) change very rapidly.

Motivation

I actually set out to learn Jena Rules using the Semantic Web Programming book as a guide. Midway through that exercise, it occurred to me that Prolog would be a cleaner and almost drop-in replacement to the rather verbose Turtle syntax. Apparently the Semantic Web community thinks otherwise, since Turtle stands for Terse RDF Triple language. I haven't actually used Prolog before this, although I've read code snippets in articles once or twice (but not recently), so the realization was almost like an epiphany.

Which Prolog?

I initially download GNU-Prolog because it was available from the yum repository, but then I decided to go with SWI-Prolog, because there is a Netbeans plugin available for it, and because it offers a Java-Prolog Interface (JPL) (haven't tried this yet). Because SWI-Prolog did not have an RPM for my AMD-64 laptop, I had to build it from source, but I did not have any problems doing that.

Learning Prolog

There are quite a few Prolog tutorials available on the Web, but most focus on trying to use it as a general-purpose programming language. Since I intended to use Prolog only for its logic programming facilities, I found the Learn Prolog Now! and Adventure in Prolog online books more suitable. The first one is based on SWI-Prolog and the second on Amzi! Prolog, but examples from both worked fine for me.

The Fact Base

A Prolog program consists of facts, rules and queries. In order to keep my fact base similar to the ontology model, I decided to model my facts as triples (isTriple/3 in Prolog, since it takes three arguments), as shown below. Each of the subject, predicate and object can either be an Atom or a Compound Term (you have to make this decision at modeling time). I've just used Atoms in my example.

```
isTriple(subject, predicate, object).
% if we want predicates to have a property such as weight, we
% can model it as a compound term as shown below:
isTriple(subject, predicate(name, weight), object).
```

I used a simple Java program to generate my initial fact base of about 500+ triples from the sample wine.rdf file. It uses Jena to parse the file and write out the facts into a flat file. Unlike my previous usage, where I tried to map inverse relationships using an Enum, this time I only consider the relationships that exist in the wine.rdf file itself, and use rules to build the inverse relations. Since my subject and object names start with upper-case, I prepended an 'a' to make it conform to Prolog's syntax rules. You can run this with a main() class or write a unit test. I used a unit test, but I am not showing this since its so trivial.

```
// Source: src/main/java/net/sf/jtmt/inferencing/prolog/Owl2PrologFactGer
package net.sf.jtmt.inferencing.prolog;

import java.io.FileWriter;
import java.io.PrintWriter;

import org.apache.commons.lang.StringUtils;

import com.hp.hpl.jena.graph.Node;
import com.hp.hpl.jena.graph.Node_URI;
import com.hp.hpl.jena.graph.Triple;
import com.hp.hpl.jena.rdf.model.Model;
import com.hp.hpl.jena.rdf.model.ModelFactory;
import com.hp.hpl.jena.rdf.model.Statement;
import com.hp.hpl.jena.rdf.model.StmtIterator;

/**
 * Reads an OWL file representing an ontology, and outputs a Prolog
 * fact base.
 */
public class Owl2PrologFactGenerator {

  private String inputOwlFilename;
  private String outputPrologFilename;

  public void setInputOwlFilename(String inputOwlFilename) {
    this.inputOwlFilename = inputOwlFilename;
  }

  public void setOutputPrologFilename(String outputPrologFilename) {
    this.outputPrologFilename = outputPrologFilename;
  }

  public void generate() throws Exception {
    PrintWriter prologWriter =
      new PrintWriter(new FileWriter(outputPrologFilename), true);
    Model model = ModelFactory.createDefaultModel();
    model.read(inputOwlFilename);
    StmtIterator sit = model.listStatements();
    while (sit.hasNext()) {
      Statement st = sit.next();
      Triple triple = st.asTriple();
      String prologFact = getPrologFact(triple);
      if (StringUtils.isNotEmpty(prologFact)) {
        prologWriter.println(getPrologFact(triple));
```

databases dbunit debugging decorator-pattern development dijkstra dimensionality-reduction distributed-computing django documentation dojo drupal dwr dynabean ec2 eclipse edb emacs embedded-databases enterprisedb etl eventbus events facets fedora feeds functors garbage-collection gdata general generics genetic-algorithms gnuplot google-web-toolkit graph gwt hadoop hbase hessian hibernate hidden-markov-model html httpclient image-processing imap indexing information-retrieval inverse-document-frequency inversion-of-control ir j2ee jaccard-similarity jackrabbit jahmm jama jatha java javaconfig javamail javascript javaspaces jax jazzy jcr jdbc jdee jdom jetlang jetty jfreechart jgrapht jlisp jms jmx journaling jpa json jsp junit jvm jwi jython k-means kettle kilim larvalabs latent-semantic-indexing lighttpd linear-algebra lingpipe linux lisp log-analysis lucene lucli luke mac-os-x machine-learning mahout map-reduce math matplotlib matrix maven maven-plugin maven2 messaging mockobjects mojo mongodb mono mule multithreading mvc mysql naive-bayes nearest-neighbor neo4j nlp nltk nosql nutch object-oriented-design object-persistence ontology opennlp oracle osworkflow owl parallel parser parser-toolkit patterns pca pelops performance persistence personalization petri-net php pos-tagging postgresql prevayler principal-component-analysis probability profiling prolog prototype pygments pylons PyLucene python qdox qt r-language rcs recommender redhat remoting rest robert-sedgewick rome rome-custom-modules roo ror rss rsync ruby-on-rails rules scala scalding scikit-learn scripting search security semantic-web shortest-path similarity simulated-annealing singular-value-decomposition smtp soap solr spark spatial spell-checking spring springmodules-jcr springmodules-lucene squid stax tag-cloud tagging tapestry taxonomy tf-idf tiles transactions trees turbogears ubuntu uima unison unit-testing unix url-rewriting urlrewrite-filter velocity version-control web web-development web-service webapp-security webwork weka windows workflow xfire xml xmlrpc xpath

Blogs I Read

```
      }
    }
    model.close();
    prologWriter.flush();
    prologWriter.close();
  }

  private String getPrologFact(Triple triple) {
    StringBuilder buf = new StringBuilder();
    Node subject = triple.getSubject();
    Node object = triple.getObject();
    if ((subject instanceof Node_URI) &&
        (object instanceof Node_URI)) {
      buf.append("isTriple(a").
      append(triple.getSubject().getLocalName()).
      append(",").
      append(triple.getPredicate().getLocalName()).
      append(",a").
      append(triple.getObject().getLocalName()).
      append(").");
    }
    return buf.toString();
  }
}
```

My output file contains the fact base in Prolog syntax. Here is a partial listing, to show you how it looks. The full source file (including the rules and the testing function, described below) is available here if you want it.

```
% Source: src/main/prolog/net/sf/jtmt/inferencing/prolog/wine_facts.pro
% ...
isTriple(aCorbansPrivateBinSauvignonBlanc,hasBody,aFull).
isTriple(aCorbansPrivateBinSauvignonBlanc,hasFlavor,aStrong).
isTriple(aCorbansPrivateBinSauvignonBlanc,hasSugar,aDry).
isTriple(aCorbansPrivateBinSauvignonBlanc,hasMaker,aCorbans).
isTriple(aCorbansPrivateBinSauvignonBlanc,locatedIn,aNewZealandRegion).
isTriple(aCorbansPrivateBinSauvignonBlanc,type,aSauvignonBlanc).
isTriple(aSevreEtMaineMuscadet,hasMaker,aSevreEtMaine).
isTriple(aSevreEtMaineMuscadet,type,aMuscadet).
isTriple(aWineFlavor,subClassOf,aWineTaste).
isTriple(aWineFlavor,type,aClass).
isTriple(aEdnaValleyRegion,locatedIn,aCaliforniaRegion).
isTriple(aEdnaValleyRegion,type,aRegion).
...
```

Adding Rules

The first step is adding the inverse relationships using Prolog rules. This is quite simple, as shown below:

```
% Source: src/main/prolog/net/sf/jtmt/inferencing/prolog/wine_facts.pro
% ...
% ---------------------------------------------------------------
%         rules to augment the generated facts.
% ---------------------------------------------------------------

% rules to generate inverse relationships where applicable
isTriple(Subject, isVintageYearOf, Object) :-
    isTriple(Object, hasVintageYear, Subject).
isTriple(Subject, regionContains, Object) :-
    isTriple(Object, locatedIn, Subject).
isTriple(Subject, mainIngredient, Object) :-
    isTriple(Object, mainIngredient, Subject).
isTriple(Subject, isFlavorOf, Object) :-
    isTriple(Object, hasFlavor, Subject).
isTriple(Subject, isColorOf, Object) :-
```

Data Recipes
1 year ago

Edwin Chen's Blog
2 years ago

FastML
1 week ago

IR Thoughts
19 hours ago

Lucid Imagination Blog
4 weeks ago

Mawazo
1 week ago

Nihil Obstat
8 months ago

No Free Hunch
2 days ago

OpenSource Connections Blog
6 months ago

Peter Norvig
3 days ago

Scale Unlimited Blog
7 months ago

Sematext Blog
14 hours ago

StreamHacker
5 months ago

Surprise and Coincidence...
1 year ago

Text & Data Mining by Practical Means
6 months ago

About me

Sujit Pal

I am a programmer interested in Semantic Search, Ontology, Natural Language Processing and Machine Learning. My programming languages of choice are Java, Scala, and Python. I love solving problems and exploring different possibilities with open source tools and frameworks.

View my complete profile

My Nerd Rating

I am nerdier than 95% of all people. Are you nerdier? Click here to find out!

Subscribe

Email address... Submit

Visitor Map

Powered By Blogger

```
    isTriple(Object, hasColor, Subject).
isTriple(Subject, isSugarContentOf, Object) :-
    isTriple(Object, hasSugar, Subject).
isTriple(Subject, isBodyOf, Object) :-
    isTriple(Object, hasBody, Subject).
isTriple(Subject, madeBy, Object) :-
    isTriple(Object, hasMaker, Subject).
isTriple(Subject, hasInstance, Object) :-
    isTriple(Object, type, Subject).
isTriple(Subject, superClassOf, Object) :-
    isTriple(Object, subClassOf, Subject).
```

Nothing fancy here, as you can see - we just create new isTriple rules by switching the subject and object around, and replacing the predicate with its inverse. These are simple examples of generating relationships algebrically from existing ones, we have slightly more complex examples later. Trying out some of these rules in the SWI-Prolog listener (AKA interactive shell in Python, or REPL in Lisp) shows that they work. Note that the last false indicates that there are no more matches for this rule.

```
sujit@sirocco:~$ pl
Welcome to SWI-Prolog (Multi-threaded, 64 bits, Version 5.6.64)
Copyright (c) 1990-2008 University of Amsterdam.
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to redistribute it under certain conditions.
Please visit http://www.swi-prolog.org for details.

?- consult('wine_facts.pro').
% wine_facts.pro compiled 0.02 sec, 109,832 bytes
true.

?- isTriple(aCongressSpringsSemillon, hasSugar, aDry).
true ;
false.

?- isTriple(aDry, isSugarContentOf, aCongressSpringsSemillon).
true ;
false.
```

I then decided to add relations which don't already exist, using slightly more complex rules (involving recursion) to generate relationships from existing ones. Here is the snippet for these rules from my wine_facts.pro file.

```
% Source: src/main/prolog/net/sf/jtmt/inferencing/prolog/wine_facts.pro
% ...
% rule to find all wines produced by a given region (region can be at any
% level, ie. country (USRegion), state (CaliforniaRegion), or location wi
% state (SantaCruzMountainsRegion). Only wines should be listed. We do th
% by ensuring that a Wine has a valid maker.
isTriple(Region, produces, Wine) :- isTriple(Region, regionContains, Wine
                                    isTriple(Wine, hasMaker, _).
isTriple(Region, produces, Wine) :- isTriple(Region, regionContains, X),
                                    isTriple(X, produces, Wine),
                                    isTriple(Wine, hasMaker, _).

% rule to find out the region for which the wine is produced. Only the
% regions should be listed. We do this by ensuring that a Region has type
% aRegion.
isTriple(Wine, producedBy, Region) :- isTriple(Region, regionContains, Wi
                                      isTriple(Region, type, aRegion).
isTriple(Wine, producedBy, Region) :- isTriple(Region, regionContains, X)
                                      isTriple(X, produces, Wine),
                                      isTriple(X, type, aRegion).
```

As before we can test these rules from the SWI-Prolog shell. However, I also built a little Prolog function that allows you to do Query-By-Example.

```
% Source: src/main/prolog/net/sf/jtmt/inferencing/prolog/wine_facts.pro
% --------------------------------------------------------------
%     simple query-by-example testing tool
% --------------------------------------------------------------
test(Subject,Predicate,Object) :- isTriple(Subject, Predicate, Object),
    tab(2), write('('), write(Subject),
    write(','), write(Predicate),
    write(','), write(Object),
    write(')'), nl, fail.
```

Running my test cases (commented out in the source file, since I could not get them to work in batch mode) in the SWI-Prolog listener returns the following (expected) results.

```
?- consult('wine_facts.pro').
% wine_facts.pro compiled 0.02 sec, 109,832 bytes
true.

?- test(aUSRegion, produces, X).
  (aUSRegion,produces,aMountEdenVineyardEstatePinotNoir)
  (aUSRegion,produces,aMountEdenVineyardEdnaValleyChardonnay)
  (aUSRegion,produces,aFormanChardonnay)
  (aUSRegion,produces,aWhitehallLaneCabernetFranc)
  (aUSRegion,produces,aFormanCabernetSauvignon)
  (aUSRegion,produces,aElyseZinfandel)
  (aUSRegion,produces,aSeanThackreySiriusPetiteSyrah)
  (aUSRegion,produces,aPageMillWineryCabernetSauvignon)
  (aUSRegion,produces,aBancroftChardonnay)
  (aUSRegion,produces,aSaucelitoCanyonZinfandel)
  (aUSRegion,produces,aSaucelitoCanyonZinfandel1998)
  (aUSRegion,produces,aMariettaPetiteSyrah)
  (aUSRegion,produces,aMariettaZinfandel)
  (aUSRegion,produces,aGaryFarrellMerlot)
  (aUSRegion,produces,aPeterMccoyChardonnay)
  (aUSRegion,produces,aMariettaOldVinesRed)
  (aUSRegion,produces,aCotturiZinfandel)
  (aUSRegion,produces,aMariettaCabernetSauvignon)
  (aUSRegion,produces,aVentanaCheninBlanc)
  (aUSRegion,produces,aLaneTannerPinotNoir)
  (aUSRegion,produces,aFoxenCheninBlanc)
  (aUSRegion,produces,aSantaCruzMountainVineyardCabernetSauvignon)
  (aUSRegion,produces,aStGenevieveTexasWhite)
false.

?- test(X, produces, aLaneTannerPinotNoir).
  (aSantaBarbaraRegion,produces,aLaneTannerPinotNoir)
  (aCaliforniaRegion,produces,aLaneTannerPinotNoir)
  (aUSRegion,produces,aLaneTannerPinotNoir)
false.

?- test(aTexasRegion, produces, X).
  (aTexasRegion,produces,aStGenevieveTexasWhite)
false.

?- test(X, produces, aStGenevieveTexasWhite).
  (aCentralTexasRegion,produces,aStGenevieveTexasWhite)
  (aTexasRegion,produces,aStGenevieveTexasWhite)
  (aUSRegion,produces,aStGenevieveTexasWhite)
false.

?- test(X, producedBy, aUSRegion).
  (aCaliforniaRegion,producedBy,aUSRegion)
  (aTexasRegion,producedBy,aUSRegion)
  (aMountEdenVineyardEstatePinotNoir,producedBy,aUSRegion)
  (aMountEdenVineyardEdnaValleyChardonnay,producedBy,aUSRegion)
  (aFormanChardonnay,producedBy,aUSRegion)
  (aWhitehallLaneCabernetFranc,producedBy,aUSRegion)
  (aFormanCabernetSauvignon,producedBy,aUSRegion)
  (aElyseZinfandel,producedBy,aUSRegion)
  (aSeanThackreySiriusPetiteSyrah,producedBy,aUSRegion)
  (aPageMillWineryCabernetSauvignon,producedBy,aUSRegion)
  (aBancroftChardonnay,producedBy,aUSRegion)
  (aSaucelitoCanyonZinfandel,producedBy,aUSRegion)
  (aSaucelitoCanyonZinfandel1998,producedBy,aUSRegion)
  (aMariettaPetiteSyrah,producedBy,aUSRegion)
  (aMariettaZinfandel,producedBy,aUSRegion)
  (aGaryFarrellMerlot,producedBy,aUSRegion)
  (aPeterMccoyChardonnay,producedBy,aUSRegion)
  (aMariettaOldVinesRed,producedBy,aUSRegion)
  (aCotturiZinfandel,producedBy,aUSRegion)
  (aMariettaCabernetSauvignon,producedBy,aUSRegion)
  (aVentanaCheninBlanc,producedBy,aUSRegion)
  (aLaneTannerPinotNoir,producedBy,aUSRegion)
  (aFoxenCheninBlanc,producedBy,aUSRegion)
  (aSantaCruzMountainVineyardCabernetSauvignon,producedBy,aUSRegion)
  (aStGenevieveTexasWhite,producedBy,aUSRegion)
false.

?- test(X, producedBy, aTexasRegion).
  (aCentralTexasRegion,producedBy,aTexasRegion)
  (aStGenevieveTexasWhite,producedBy,aTexasRegion)
false.

?- test(aLaneTannerPinotNoir, producedBy, X).
  (aLaneTannerPinotNoir,producedBy,aSantaBarbaraRegion)
  (aLaneTannerPinotNoir,producedBy,aCaliforniaRegion)
  (aLaneTannerPinotNoir,producedBy,aUSRegion)
false.

?- test(aStGenevieveTexasWhite, producedBy, X).
  (aStGenevieveTexasWhite,producedBy,aCentralTexasRegion)
  (aStGenevieveTexasWhite,producedBy,aTexasRegion)
  (aStGenevieveTexasWhite,producedBy,aUSRegion)
false.
```

Conclusion

I found this article (PDF) describing an attempt to model OWL rules using Prolog, so perhaps this idea is not as novel as it seemed to me at first. Prolog uses backward inferencing, which means that the rule based facts are recomputed on demand, rather than at the point of being asserted into the factbase. For a query-heavy system, which most rule based systems tend to be, this can have an impact on performance. But I think an application built around Prolog's rule engine can get around this by identifying a fact based on its origin, and generating and caching rule based facts at the point of assertion. If a rule is dropped or modified, the facts based on that rule could be recomputed and cached automatically.

In terms of simplicity of syntax alone, I think a Prolog based rule definition system would be a welcome addition to the Semantic Web Programmer's toolkit. The pattern-based query by example I have described is also likely to be much simpler and easier to use than the more imperative SPARQL query language used to query OWL ontologies.

However, I *do plan* to learn how to build rules using the tools and languages in the Jena framework, just because it is what I am more likely to use in a typical Semantic Web development environment.

Posted by Sujit Pal at 8:18 AM

Labels: ontology, prolog, rules

Share/Save:

4 comments (moderated to prevent spam):

Chris said...

There is a package for SWI-Prolog called Thea that allows for translation of OWL2 ontologies to rules, as well as limited backward chaining strategies. See:

http://github.com/vangelisv/thea

You can also do reverse translations for a limited subset of prolog. See:

http://blipkit.wordpress.com/2009/06/19/translating-between-logic-programs-and-owlswrl/

7/22/2009 9:10 PM

Sujit Pal said...

Thank you Chris, I took a quick look at Thea, will take a more detailed look later.

7/23/2009 6:19 PM

Anonymous said...

But Mr Cris How to use THEA. I am not getting anything. I am a biginner

11/15/2012 10:24 AM

Sujit Pal said...

Hi, you may want to post this query on Chris's blog.

11/15/2012 5:24 PM

Post a Comment

Links to this post

Create a Link

Newer Post Home Older Post

Subscribe to: Post Comments (Atom)