CWM

Closed World Machine

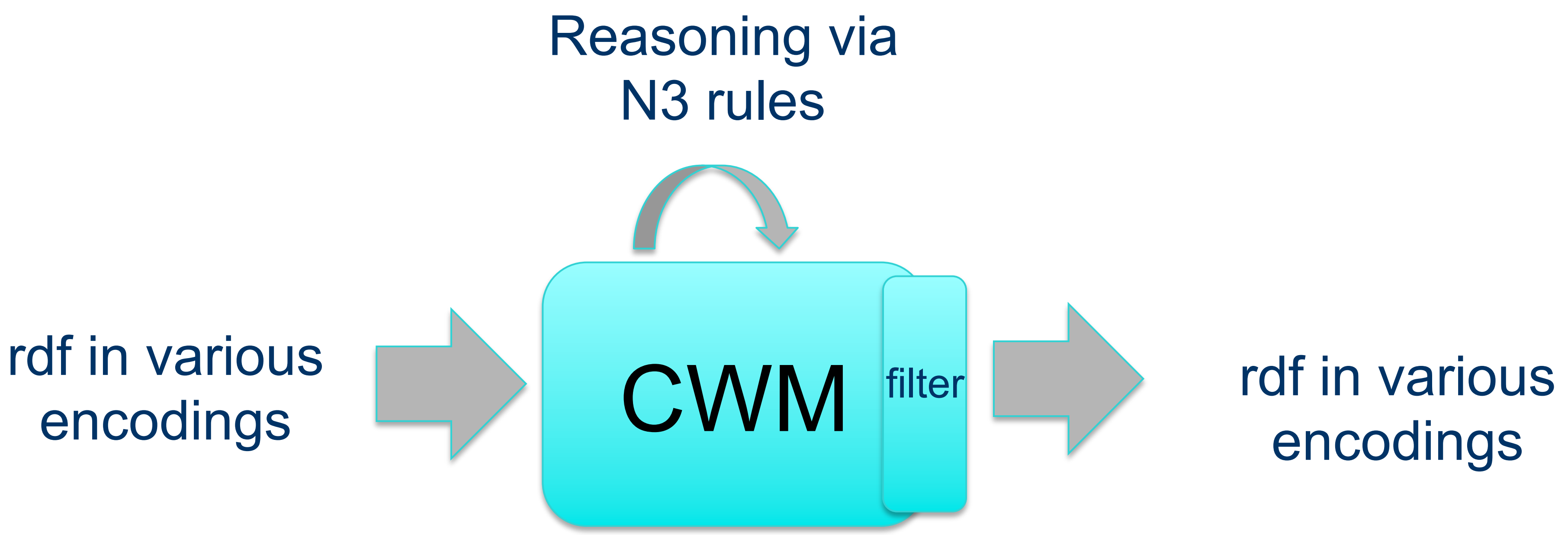
CWM Overview

- CWM is a popular Semantic Web program that can do the following tasks
  - Parse and pretty-print several RDF formats: XML RDF, Notation3, and Ntriples
  - Store triples in a queryable triples database
  - Perform inferences as a forward chaining inference engine
  - Perform builtin functions such as comparing strings, retrieving resources, all using an extensible builtins suite
- CWM was written in Python by Tim Berners-Lee and Dan Connolly of the W3C

What's CWM good for?

- CWM is good for experimenting with RDF and RDFS and some OWL
- CWM's rule based reasoner can't cover all of OWL
- It's good as a Unix tool that you can call from the command line
- rdfs:seeAlso
  - http://infomesh.net/2001/cwm/
  - http://w3.org/2000/10/swap/doc/Processing

CWM in a Nutshell



Some alternative libraries

If you want to play with RDF and RDFS from a programming language, you might check out some of these:

- Redland RDF Libraries (C): http://librdf.org/
- Jena (Java): http://jena.sourceforge.net/
- Sesame (Java): http://www.openrdf.org/
- RDFLib (Python): http://rdflib.net/
- SWI (Prolog): http://www.swi-prolog.org/web
- Wilbur (Lisp): http://wilbur-rdf.sourceforge.net/

CWM command line

- Example: cwm --rdf foo.rdf --n3 > foo.n3
- Args are processed left to right (except for flags --pipe and –help
- Here's what happens:
  - Switch to RDF/XML input-output format
  - Read in foo.rdf (use a filename or URI) and add triples to store
  - Switch to --n3 input-output format
  - Output triples in store to stdout in N3
  - Unic redirect captures output in foo.ne

On N3 and Turtle

- The N3 notation was invented by Tim Berners Lee
- It's not a standard, but a large subset, Turtle, is (almost) a standard
- What's in N3 but not in Turtle
  - Representing inference rules over RDF triples
  - A compact syntac for reification
  - Some other bits
- The rules part is most useful
  - It's been supplanted in part by SPARQL
  - And by RIF (Rule Interchange Formalism)

Reasoning using N3 Rules

- N3 has a simple notation for Prolog like rules
- These are represented in RDF, of course, and can read these into CWM just like a data file
- Command line args tell CWM to reason
  - **--apply=foo :** read rules from foo, apply to store, adding conclusions to store
  - **--rules :** apply once the rules in the store to the store, adding conclusions to the store
  - **--filter=foo :** apply rules in foo to the store, REPLACING the store with the conclusions
  - **--think :** apply rules in store to the store, adding conclusions to the store, iteratively until no more new conclusions are made

N3 facts and rules

- :Pat owl:sameAs :Patrick .
- :Man rdfs:subclassOf :Human .
  :YoungMan rdfs:subclassOf :Man .
- :has_father rdfs:domain :Human; s:range :Man .
  :Sara :has_father :Alan .
- { ?x :has_parent ?y } => { ?y :has_child :?x } .
- {?x :has_parent ?y. ?y :has_brother ?z}
  => {?x :has_uncle ?z} .
- { :thermostat :temp ?x. ?x math:greaterThan "70" } => { :cooling :power "high" } .

Implications in logic

- In logic, an implication is a sentence that is either true or false
  - forall X man(x) => mortal(x)
- If we believe an implication to be true, we can use it to derive new sentences that must be true from others we believe true
  - man(socrates) therefore mortal(socrates)
- This is the basis for the rule based reasoning systems
  - Prolog, Datalog, Jess, etc.

Quantifiers

- In first order logic, we have two quantifiers, for all (∀) and exists (∃)
  - ∀x ∃y has_child(x, y) => is_parent(x)
  - For all x, if there exists a y such that x has_child y, then x is a parent or
  - X is a parent if X has (at least) one child
  - You only need to find one child to conclude that someone is a parent
- Variables (e.g., x and y) range over all *objects* in the universe, but for KB systems, we can narrow this to objects mentioned in the KB

Variables in rules implicitly quantified

- Most rule-based systems don't use explicit quantifiers
- Variables are implicitly quantified as either ∀ or ∃ using the following
  - All variables in the rule's conclusion are universally quantified
  - All variables appearing *only* in the premise are existentially quantified
- Thus foo(a,b) => bar(b) is interpreted as ∀b ∃a foo(a,b)) => bar(b)

Variables in rules implicitly quantified

- To see why this is a reasonable design decision for a rule language, consider
  - $\forall x \ \forall y$ has_child(x, y) => is_parent(x)
- What does this mean?
  - "X is a parent if we can prove that X has *every object* in our universe as a child"
- Such rules are not often useful
- Many rule languages do have ways to express them, of course

Reasoning: Forward and Backward

- Rule based systems ten to use one of two reasoning strategies (and some do both)
  - Reasoning *forward* from known facts to new ones (find all people who are parents; is Bob among them?)
  - Reasoning *backward* from a conclusion posed as a query to see if it is true (Is Bob a parent)
- Each has advantages and disadvantages which may effect its utility in a given use case
- CWM uses a forward reasoning strategy
  - Principally because we often want to compute all RDF triples that follow from a given set (i.e., find the deductive closure)

N3 Rules

- An N3 rule has a *conjunction* of triples as a premise and a *conjunction* as a conclusion
- Eg: 2nd element of a triple is always a property

  { ?S ?P ?O. } => { ?P a rdf:Property. }
- Eg: The meaning of rdfs:domain

  { ?S ?P ?O. ?P rdfs:domain ?D.} => { ?O a ?D. }
- Variables begin with a ?.
- Every variable in the conclusions must appear in the premise
- Each way to instantiate triple patterns in the pre-mise with a set of triples in the KB yields new facts

Note: limited negation & disjunction

- Disjunction in the premise can be achieved using several rules
  - { ?S :foo1 ?0.} => { ?S :bar ?O.}
  - { ?S :foo2 ?0.} => { ?S :bar ?O.}
- No disjunction is allows in the conclusion
- No general negation is allowed (there is a very limited kind of negation)
- Negation and disjunction are supported in other ways in OWL and RIF and in other reasoners

N3 rules use cases

- Use N3 rules to implement the semantics of RDF, RDFS, and OWL vocabularies
  - See rdfs-rules.n3
  - See owl-rules.n3
- Use N3 rules to provide domain/application specific rules
  - See gedcom-relations.n3

A simple example

```
% more simple1.n3

# A simple example

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix : <#> .

:john a foaf:Person;
   foaf:name "John Smith";
   foaf:gender "Male";
   foaf:name "John Smith" .
```

Invoking CSM (1)

```
% cwm simple1.n3
# Processed by Id: cwm.py,v 1.197 2007/12/13 15:38:39 syosi Exp
# using base file:///Users/finin/Sites/691s13/examples/n3/simple1.n3
#  Notation3 generation by notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp
#   Base was: file:///Users/finin/Sites/691s13/examples/n3/simple1.n3
@prefix : <#> .

    :john a <http://xmlns.com/foaf/0.1/Person>;
        <http://xmlns.com/foaf/0.1/gender> "Male";
        <http://xmlns.com/foaf/0.1/name> "John Smith" .
#ENDS
```

Invoking CSM (2)

```
n3> cwm -n3=/d simple1.n3
# Processed by Id: cwm.py,v 1.197 2007/12/13 15:38:39 syosi Exp
# using base file:///Users/finin/Sites/691s13/examples/n3/simple1.n3
#  Notation3 generation by  notation3.py,v 1.200 2007/12/11 21:18:08 syosi Exp
#   Base was: file:///Users/finin/Sites/691s13/examples/n3/simple1.n3

    @prefix foaf: <http://xmlns.com/foaf/0.1/> .

    <#john>     a foaf:Person;
         foaf:gender "Male";
         foaf:name "John Smith" .
```

Some useful CWM flags

- CWM command has a lot of flags and switches
- Do cwm --help to see them
- Here are a few

--rdf Input & Output ** in RDF/XML insead of n3 from now on

--n3 Input & Output in N3 from now on. (Default)

--n3=flags Input & Output in N3 and set N3 flags

--ntriples Input & Output in NTriples (equiv --n3=usbpartane -bySubject -quiet)

--apply=foo Read rules from foo, apply to store, adding conclusions to store

--think as -rules but continue until no more rule matches (or forever!)

--think=foo as -apply=foo but continue until no more rule matches (or forever!)

--data Remove all except plain RDF triples (formulae, forAll, etc)

--help print this message

RDFS in N3 (1)

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
…
rdfs:comment rdfs:domain rdfs:Resource; rdfs:range rdfs:Literal.
rdfs:domain rdfs:domain rdf:Property; rdfs:range rdfs:Class.
rdfs:label rdfs:domain rdfs:Resource; rdfs:range rdfs:Literal.
rdfs:range rdfs:domain rdf:Property; rdfs:range rdfs:Class.
rdfs:seeAlso rdfs:domain rdfs:Resource; rdfs:range rdfs:Resource.
rdfs:subClassOf rdfs:domain rdfs:Class; rdfs:range rdfs:Class.
rdfs:subPropertyOf rdfs:domain rdf:Property; rdfs:range rdf:Property.
rdf:type rdfs:domain rdfs:Resource; rdfs:range rdfs:Class.
…
```

RDFS in N3 (2)

```
{?S ?P ?O} => {?P a rdf:Property}.
{?S ?P ?O} => {?S a rdfs:Resource}.
{?S ?P ?O} => {?O a rdfs:Resource}.

{?P rdfs:domain ?C. ?S ?P ?O} => {?S a ?C}.
{?P rdfs:range ?C. ?S ?P ?O} => {?O a ?C}.

{?Q rdfs:subPropertyOf ?R. ?P rdfs:subPropertyOf ?Q}
    => {?P rdfs:subPropertyOf ?R}.
{?P rdfs:subPropertyOf ?R. ?S ?P ?O} => {?S ?R ?O}.

{?A rdfs:subClassOf ?B. ?S a ?A} => {?S a ?B}.
{?B rdfs:subClassOf ?C. ?A rdfs:subClassOf ?B}
    => {?A rdfs:subClassOf ?C}.
```

Demonstration

- Install cwm
- Download files in the n3 examples directory http://cs.umbc.edu/courses/graduate/691/spring13/01/examples/n3/

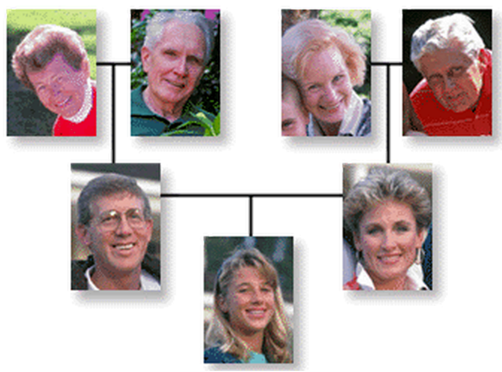
HW3

UMBC CMSC 491/691 The S

www.csee.umbc.edu/courses/graduate/691/spring13/01/hw/hw3/

The Semantic Web

Home ·Syllabus · Schedule · HW · Exams · Notes · Code · Examples · Resources · Discussion · Blog · Email



Assignment Three

Experimenting with RDFS using N3

Due 28 February 2013

N3 is a notation for RDF that is easier for people to read and write than XML/RDF. N3 also supports a simple syntax for rules that allows us to define rules to implement the meaning of RDF and (most of) OWL as well as other domain specific reasoning over RDF data. Turtle is a simplified, RDF-only subset of N3 that is beging developed as a possible W3C recommendation.

CWM is a simple reasoner implemented in Python that you can use to experiment with both N3 and reasoning over RDF content. You can download and install cwm on your own computer or use it on the CSEE linux systems. CWM is a python program, so you may need to install python if you are running WIndows. on GL, you can use the version I have installed in my files. Adding one of the following to your .cshrc (if your shell is tcsh) or .bashrc (for bash).

> ***for csh and tcsh:*** `alias cwm "/afs/umbc.edu/users/f/i/finin/pub/cwm"`
> ***for bash:*** `alias cwm="/afs/umbc.edu/users/f/i/finin/pub/cwm"`

CWM uses a straightforward forward chaining reasoning approach. It can import or export semantic web documents from local files or URLs in either N3 or RDF/XML. Some of these files can be rules (in N3) that define the semantics of RDFS and/or OWL. The rules are applied and the complete set of triples can be output.

Summary

- CWM is a relatively simple program that lets you manipulate and explore RDF and Semantic Web technology
- It's limited in what it can do and not very efficient
- But useful and "close to the machine"
- Written in Python
- There are related tools in Python, see rdflib
- And lots more tools in other languages

genesis

```
# A simple example of family relations
    using the gedcom vocabulary.

@prefix gc: <http://www.daml.org/
    2001/01/gedcom/gedcom#>.
@prefix log: <http://www.w3.org/2000/10/
    swap/log#>.
@prefix owl: <http://www.w3.org/2002/07/
    owl#>.
@prefix : <#> .
# data from the Bible in GEDCOM form
:fam1 a gc:Family.

:Able gc:sex gc:Male;
  gc:givenName "Able";
  gc:childIn :fam1;
  owl:differentFrom :Cain.
```

```
:Cain gc:sex gc:Male;
  gc:givenName "Cain";
  gc:childIn :fam1;
  owl:differentFrom :Able.

:Adam gc:sex gc:Male;
  gc:givenName "Adam";
  gc:spouseIn :fam1;
  owl:differentFrom :Eve.

:Eve gc:sex gc:Female;
  gc:givenName "Eve";
  gc:spouseIn :fam1;
  owl:differentFrom
```