Embedding semantic markup in LaTeXand Markdown documents

Will Ware

24 February 2016

1 Preliminaries

1.1 Overview

My goal here is to figure out what I call a *machine-tractable* scientific literature: a nice clean way to embed machine-processable semantic information in the source of LaTeX and Markdown documents, such software pre-processors can extract the information in machine-tractable form and can also generate (ideally publication-ready) HTML or PDF output.

This idea is part of a larger group of ideas that I have been thinking about since around 2010.

> ...to formulate a linked language of science that machines can understand. Publish papers in formats like RDF/Turtle or JSON or JSON-LD or YAML. Link scientific literature to existing semantic networks (DBpedia, Freebase, Google Knowledge Graph, Linked-Data.org, Schema.org etc). Create schemas for scientific domains and for the scientific method (hypotheses, predictions, experiments, data). Provide tutorials, tools and incentives to encourage researchers to publish machine-tractable papers. Create a distributed graph or database of these papers, in the role of scientific journals, accessible to people and machines everywhere. Maybe use Stackoverflow as a model for peer review.

My immediate concern is with the task of comfortably embedding such formats in LaTeX or Markdown source. I say *comfortably* because I hope for eventual wide adoption by the authors of scientific literature, and that is unlikely if semantic markup presents any significant additional burden to publication. Consequently the syntax for such markup must be simple and its meaning obvious.

In particular it should not be necessary to specify the same idea twice, once in English and again in machine-tractable form. As with mathematical equations, the machine-tractable representation should be readable and expressive to the human audience as well as the machine.

1.2 Machine representations

There have been huge advances in machine learning in recent years, fueled by large investments from Google, Facebook, and other organizations. I have not kept up with this work as well as I would like, and no doubt my ideas in this area will seem antiquated to those who have.

I have tried not to commit this approach to any particular representation, but I've tended to use RDF/Turtle because first-order logic doesn't sound like a bad place to start for machine reasoning about science. The notion of specifying information or knowledge as a semantic network seems to me a useful one.

1.3 Notation

These considerations led me to adopt a literate programming approach, borrowing notationally from Norman Ramsey's noweb. I am not the first to rewrite noweb in Python.

In literate programming, one writes a program in small pieces, discussing each in turn. Likewise, a semantic network can be specified in pieces, with discussion. Here the first line gives a name to this piece so that it can be referenced elsewhere. The remaining lines are in the RDF/Turtle language. They tell us that, in the world of Marvel comics superheros, Spiderman is a person named *Spiderman* who has an enemy (the Green Goblin, to be discussed shortly).

```
<<semantic info about spiderman>> =
<#spiderman>
    a foaf:Person ;
    foaf:name "Spiderman" ;
    rel:enemyOf <#green-goblin> .
```

Terms like *foaf:Person* refer to FOAF, or *friend of a friend*, a library of semantic relationships between people. Another similar library is Relationship, which include the *rel:enemyOf* term.

The components of a semantic network are triplets. Blah blah blah triplets.... semicolons... commans... period.

The Green Goblin is another character in the Marvel comics universe, not so different from Spiderman in some ways.

```
<<semantic info about the green goblin>> =
<#green-goblin>
    a foaf:Person ;
    foaf:name "Green Goblin" ;
    rel:enemyOf <#spiderman> .
```

Finally we have a wrapper for the two pieces of semantic network above. Here we provide referents for the prefixes *foaf* and *rel*, and other prefixes that are commonly used in RDF.

```
<<rdf>> =
@base <http://example.org/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rel: <http://vocab.org/relationship/> .
<<semantic info about spiderman>>
<<semantic info about the green goblin>>
```

References: *semantic info about spiderman*, *semantic info about the green goblin*

2 The bigger picture

2.1 Inference

In RDF terms, *inference* can be considered a synonym for *deduction*. Mechanically, it's a process whereby some old RDF triples go in, and some new RDF triples come out. In practice this is done with software called a *reasoner*, which is given a set of rules defining what constitutes valid inference.

Google *semantic reasoner* or *RDF reasoner* to learn more about reasoners.

Inference rules are often written using languages like OWL which capture some way of thinking about knowledge. For instance, OWL includes a lot of operators from set theory.

2.2 Closures

By *closure* I mean epistemic closure, not the computer science kind. A subset of epistemic closure is mathematical closure.

An example of mathematical closure is the fact that integers are closed under addition. You can add any two integers and you'll always get an integer result.

An epistemic closure is basically a complete system of thought: given a set of axioms, you've done all the reasoning to arrive at all possible conclusions, and that whole set of axioms and conclusions is your state of knowledge.

An RDF closure is the RDF subset of that, a complete set of mutually inferrable RDF triples. Any triple you can arrive at by performing inference on the existing triples is another existing triple. There is no further inference left to be done.

An RDF closure exists in the context of a set of axioms and inference rules.

In this attempt to trivialize science to what is representable in RDF, a state of scientific knowledge is represented by an RDF closure, which is in turn determined by its axioms and inference rules. It is may never be necessary to enumerate the entire contents of the closure.

2.3 Hypotheses, and Karl Popper

Assuming you've got a RDF closure representing the state of your scientific knowledge, the next step is to propose a new hypothesis, which means adding one or more new axioms. Then you infer whatever you can to arrive at the consequences of your hypothesis. Hopefully some of these will be empirically testable, and in that event you design experiments and run them to see if any of the predictions are empirically falsifiable.

If a prediction is falsified then that hypothesis is falsified, at least in the context of the existing set of hypotheses. Then you have a whole epistemological can of worms, because maybe it's only incompatible with one previous hypothesis, and maybe that one previous hypothesis might not be true. Because at no point are hypotheses *proven* to be true, they have only not yet been falsified.

Assuming for sake of argument that the list of inference rules is essentially constant...

The current set of axioms/hypotheses could be kept in a git repository, and when you want to test a new hypothesis you create a branch in the repository. That branch doesn't get merged into master until you've got a high level of confidence in the new hypothesis.

Try to come up with some meaningful numerical measure of confidence in the new hypothesis and include metadata for that in the branch. If the master branch advances significantly while the new hypothesis is under study, it would make sense to merge master back into the hypothesis branch.

2.4 Designing experiments

Initially, it probably makes sense for this machine to periodically consult a human and say

> Here are a bunch of hypotheses I've generated, starting from where the master branch was at time T. For each hypothesis, I've listed several predictions. You, the human, should decide which of these to pursue, and you should test the predictions empirically because I don't yet know how to design and run experiments. Please let me know which hypotheses are non-starters, and which predictions are a-priori known to fail.

Over time, the human will probably see opportunities for automation in (a) pruning unproductive hypotheses and predictions, and (b) turning predictions into experiments. As that happens, the machine can handle progressively larger portions of the process.

Ross King's Adam robot is an existence proof that this can be done in a non-trivial way.

2.5 Starting small

Maybe begin with experiments that can be done entirely on a computer.

- Number theories, e.g. conjectures about prime numbers
- Solving mazes and other puzzles.
- Mice? Drosophila? Bacteria or yeast in Petri dishes?

I'm afraid that exhausts my imagination. But I think well-designed puzzles could be a fruitful area, and could give me something tangible to work on until I figure out how to work on real scientific problems.

Yeast in Petri dishes is how Ross King got started, maybe that's the way to go.