DRS: A Set of Conventions for Representing Logical Languages in RDF

Drew McDermott

January 12, 2004

1 The DRS Notation

RDF, and notations based on it such as OWL, are not very good at representing arbitrary propositions, which are necessary for many applications, including the representation of processes and plans. There are various official ("OWL rules") and semiofficial ("RuleML") efforts to solve this problem. DRS is similar in spirit to the OWL Rules system, but different in detail. (For further remarks, see section 3, below.)

The key idea behind DRS is to describe a formula as an object. You can think of this as an alternative to reification, or as reification done right, depending on one's understanding of the slippery term "reification." An RDF processor must then take such a description as an assertion that the rule is true, if it occurs in an appropriate context. For example, in OWL-S, an appropriate context would be the precondition or effect of a step in a process.

The building blocks of a formula are descriptions, terms and atomic formulas. An atomic formula is represented in the XML serialization of RDF thus:

```
<drs:Atomic_formula>
   <rdf:subject>T_s</rdf:subject>
   <rdf:predicate>P</rdf:predicate>
   <rdf:object>T_b</rdf:object>
</drs:Atomic_formula>
```

where P is a predicate, T_s is a term,and T_b is a term if P is binary. If P is not binary, we have to be more devious about T_b, as we will describe shortly. We assume that the namespace `rdf` is the usual one for RDF (URI: `http://-www.w3.org/1999/02/22-rdf-syntax-ns`). We use the standard tags for reifying RDF expressions. The namespace `drs` is for vocabulary items in our formula-representation system (URI: `http://www.cs.yale.edu/homes/dvm/daml/drs-onto040112.owl#`). "DRS" stands for "Declarative RDF System."

In the simple cases, the objects P, T_s, and T_b of an atomic formula are resources, just as one would expect. For example, to assert that that Tony Blair's address is 10 Downing Street, one might write

```
<drs:Atomic_formula>
   <rdf:subject resource="...URI for Blair"/>
   <rdf:predicate resource="...URI for address"/>
   <rdf:object>
      <adr:Address>
         <adr:number>10</adr:number>
         <adr:street>Downing Street</adr:street>
         <adr:city resource="...URI for London..."/>
         ...
      </adr:Address>
   </rdf:object>
</drs:Atomic_formula>
```

Note that as far as RDF is concerned, `Atomic_formula` is just another class. Its members are formulas. It is described by an OWL ontology (located at the same URI
`http://www.cs.yale.edu/homes/dvm/daml/drsonto040112.owl`
as the one we use for the `drs` namespace). So technically all we are doing here is *describing* formulas, with the unstated convention that any formula whose description occurs at the top level of a document is considered asserted by that document. A human or computational reader of the notation that doesn't know the convention will treat the descriptions as mere descriptions (and pretty useless ones at that).

If a predicate has arity > 2, its `rdf:object` slot is filled by a a *term sequence*, represented as an OWL list of all arguments after the first. We use the same idea for representing nonatomic terms. For example, suppose we have a function `person-name` that takes three arguments, the family name, given name, and middle initial(s). In predicate calculus, this might yield the term `person-name("Musharraf", "Shlomo", "")`. In our RDF serialization, this term would be written

```
<drs:Functional_term>
   <drs:term_function resource="....#person-name"/>
   <drs:term_args rdf:parseType="Collection">
      <drs:Literal>
         <drs:litdefn rdf:datatype="&xsd;string">Musharraf</drs:litdefn>
      </drs:Literal>
      <drs:Literal>
         <drs:litdefn rdf:datatype="&xsd;string">Shlomo</drs:litdefn>
      </drs:Literal>
      <drs:Literal>
         <drs:litdefn rdf:datatype="&xsd;string"></drs:litdefn>
      </drs:Literal>
   </drs:term_args>
</drs:Functional_term>
```

(The `Literal` class and its associated property `litdefn` allow us to treat literals as individuals. OWL Full allows us for this metaphysically, but not syntactically. As long as we're making invisible extensions to RDF/OWL, we might as well introduce this convenience.)

In case you're wondering why we represented one object as an `Address` *description* and another as a `person-name` *term,* the answer is that you can do whichever seems more natural. The only caveat is that any properties of an object described using RDF are actually asserted, so if you describe a "hypothetical" object of some kind, you have to realize that RDF treats it as a description of an actual object. In other words, if you give Santa Claus's address as 0000 North Pole Drive, then as far as RDF is concerned you have asserted that there is such a roadway and a 0th location along its length. This property of RDF is why we have to resort to a coding scheme in the first place.

If an atomic formula has exactly one argument, the `object` slot is occupied by a special object called `null` from the DRS namespace. If it has zero arguments, both the `subject` and `object` slots are both `null`.

Above the level of atomic formulas we have connectives, which are just classes of formulas with names `Implies`, `And`, `Or`, and `Not`. Formulas of classes `And` and `Or` have a single property `conn_args` which is a `Term_seq` of the arguments. Formulas of class `Implies` have two properties, `antecedent` and `consequent`. Formulas of class `Not` have a single property `negated`. One can easily add more connectives if they are needed.

Variables in this system are treated as belonging to the class `Variable` from SWRL, the OWL rules language. Variables (may) have `names`, which link them back to their source form. They also have `rdf:IDs` that distinguish different variables with the same name. Variables are normally scoped by quantifiers (or other binding mechanisms), so name clashes are not a problem, but in RDF nodes have document scope.

Variables are bound with quantifiers, which are objects with two properties: the `bound_vars` (an OWL list of variables) and the `body` (a formula). There are two quantifier classes, `Forall` and `Exists`, both subclasses of `Binder`. You can add new quantifiers-like entities, such as λ, by subclassing `Binder`. You can even do without quantifiers, and use Prolog-style conventions for free variables.

Example: $\forall xy($ `loves`$(x,y) \supset$ `loves`$(y,x))$ would be represented as

```
<drs:Forall>
   <drs:bound_vars>
      <drs:Var_bag rdf:parseType="Collection">
         <swrl:Variable ID="v1" name="x"/>
         <swrl:Variable ID="v2" name="y"/>
      </drs:Var_bag>
   </drs:bound_vars>
   <drs:body>
      <drs:Implies>
         <drs:antecedent>
            <drs:Atomic_formula>
               <rdf:predicate
                  resource="http://human.rels.org/term#loves"/>
               <rdf:subject resource="#v1"/>
               <rdf:object resource="#v2"/>
            </drs:Atomic_formula>
         </drs:antecedent>
```

```
            <drs:consequent>
               <drs:Atomic_formula>
                  <rdf:predicate
                     resource="http://human.rels.org/term#loves"/>
                  <rdf:subject resource="#v2"/>
                  <rdf:object resource="#v1"/>
               </drs:Atomic_formula>
            </drs:consequent>
         </drs:Implies>
      </drs:body>
   </drs:Forall>
```

As formulas get larger and more deeply nested, they tend to wander annoyingly far to the right. Fortunately, we can fix that by moving subformulas out and making reference to them. (The cure for RDF syntax is more RDF syntax.)

2 Reference Manual, Abstract Syntax, Formal Semantics

Not really.

The abstract syntax of DRS is just the abstract syntax of any formal language you like. Ditto the formal semantics.

DRS is more a set of conventions and supporting ontology for expressing formal languages in RDF. The ontology
`http://www.cs.yale.edu/homes/dvm/daml/drsonto040112.owl`
is essentially the infrastructure required to represent formal languages, plus the grammar of the archetype, typed first-order logic. If you want to add to this grammar, it's easy to do. To add a variable-binding operator, just mimic `Forall` and make the resulting class a subclass of `Binder`. If you want to add a connective, chances are it looks like `And` or `Implies`; if not, make your own up and make it a subclass of `Connective_formula`.

3 Comparison with OWL et al.

The DRS notation is virtually identical in spirit to the OWL Rules language SWRL
`http://www.daml.org/2003/11/swrl/`.
I believe OWL Rules to be an ancestor of the standard that will finally emerge in this area, and so have tried to incorporate SWRL vocabulary into DRS wherever possible.

Unfortunately, the overlap between SWRL and DRS is currently rather small, because of some basic differences in the philosophy behind the two systems. Systems like SWRL and RuleML
`http://www.dfki.uni-kl.de/ruleml/indtd0.8.html`

are oriented toward representing "rules," while DRS is oriented toward representing propositions. I put quotes around the word "rules" because it means different things to different people. What it means in SWRL is Prolog-style Horn clauses. (RuleML is more "eclectic.") This means that variables have no explicit binders, and are implicitly universally quantified. The antecedent of a clause normally contains only atomic formulas; in SWRL nothing else is allowed.

Such differences are why we can't identify `swrl:Imp` and `drs:Implies`, which seemingly mean the same thing. `swrl:Imp` has two pieces, a `_head` and a `_body`, each a list of "atoms," which are a sort of atomic formula mimicking the assertions you can make in OWL. `drs:Implies` has two pieces, too, an `antecedent` and a `consequent`, each a formula. Technically, we could still identify `swrl:Imp` and `drs:Implies`, identifying the `antecedent` as the conjunction of the atoms in `_body`, but there's really no point. If at some point someone wants to mix SWRL and DRS notations, they can just declare that `swrl:Imp` is a `drs:Connective_formula`.

One problem with both SWRL and DRS is that such notations relegate RDF to a subordinate role; it's not really being used the way it's supposed to be. It has even been suggested that other vehicles would do just as well as RDF for the purpose of including more expressive notations in RDF documents. In particular, Bijan Parsia (personal rumor) has suggested that we could just quote a predicate-calculus formula as a string. Instead of

```
<Or>
   <conn_args rdf:parseType="Collection">
      <Atomic_formula>
         <rdf:predicate
            resource="http://human.rels.org/term#loves"/>
         <rdf:subject resource="#she"/>
         <rdf:object resource="#me"/>
      </Atomic_formula>
      <Not>
         <negated>
            <Atomic_formula>
               <rdf:predicate
                  resource="http://human.rels.org/term#loves"/>
               <rdf:subject resource="#v2"/>
               <rdf:object resource="#v1"/>
            </Atomic_formula>
         </negated>
      </Not>
   </conn_args>
<Or>
```

we could just write this

```
<logic>(or (loves she me) (not (she loves me)))</logic>
```

which is certainly more concise.

The arguments against using strings are pretty clear: Staying within RDF allows a processor to determine that a formula refers to a resource, even when it can't tell or doesn't care what it says. An XML/RDF/OWL syntax checker will do useful things with DRS notation, checking that the parens balance, that only formulas occur inside connectives, and so forth. Finally, the extra conciseness of strings is certainly attractive, but the advantage shrinks when we use non-XML-based versions of RDF.

Acknowledgements: The basis for the approach advocated here is described in

> Drew McDermott and Dejing Dou 2002 Representing Disjunction and Quantifiers in RDF
> `http://www.cs.yale.edu/homes/dvm/papers/McDermottDou02.pdf`
> Representing Disjunction and Quantifiers in RDF. *Proc. Int'l Semantic Web Conference*(available at `http://www.cs.yale.edu/homes/-dvm`).

That paper uses the N3 notation instead of the XML serialization used here.

The paper acknowledges contributions from many other people to DRS. Mark Burstein is mentioned, but it should be noted that since the publication of that paper he has continued to contribute ideas to DRS. The development of a logic dialect, Web-PDDL, for the web, and the implementation of the translator PDDOWL
`http://www.cs.yale.edu/homes/dvm/daml/pddl_daml_translator1.html`
from DRS to Web-PDDL, are the work of Dejing Dou and Peishen Qi.