# Semantic Processing of the Semantic Web*

Kunal Patel and Gopal Gupta

Applied Logic, Programming-Languages and Systems (ALPS) Laboratory
Department of Computer Science
University of Texas at Dallas
{kkp025000,gupta}@utdallas.edu

**Abstract.** We develop a semantics based approach to process information on the semantic web. We show how Horn logic can be used to denotationally capture the semantics of mark-up languages designed for describing resources on the semantic web (such as RDF). The same approach can also be used to specify the semantics of query languages for the semantic web. The semantics of both the resource description languages and the query languages are executable and when put together can be used to compute answers to semantic web queries. The main advantage of this semantic-based approach to processing the semantic web is that these executable semantics can be developed extremely quickly. Thus, as the semantic web mark-up languages evolve rapidly, their implementations can be developed at the same pace. In this paper, we present our approach based on denotational semantics and Horn logic. Our approach is quite general, and applicable to any description format (XML, RDF, DAML, etc.), though in this paper we illustrate it via RDF (Resource Description Framework).

## 1 Introduction

With increased importance of the Web in our daily lives and national economy, researchers have been looking for ways to make the WEB documents more expressive. That is, designing ways to mark-up WEB documents so that more information can be automatically inferred from them compared to what is possible with HTML. These efforts have resulted in XML (the eXtensible Mark Language) family of notation. While XML allows one to go beyond HTML, it does not take one quite far enough, since the meaning of various tags in a particular XML have to still be agreed upon. The Semantic Web [2] effort goes beyond XML by including metadata information in the document itself.

A problem with any interchange format for the semantic web is that it has to be turned into an "executable" notation, so that new information that is logically implied by the information given in the document can be automatically inferred (computed). Turning a mark-up language into an "executable" entity usually requires writing a compiler like program that maps the language to a

* Authors are partially supported by US NSF grants CCR 99-00320, CCR 98-20852, EIA 01-09347, INT 99-04063, and the US Environmental Protection Agency.

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 80–95, 2003.

notation whose semantics is mathematical; information that is implied can then be inferred automatically from this mathematical semantics.

In this paper, we develop a systematic framework for rapidly translating semantic web formats to notations that are executable. Our framework relies on Horn logic and denotational semantics. Essentially, the denotational semantics of the mark up notation is written in Horn logic. If the semantics is executable, the *denotation* of a document (i.e., the meaning assigned to the document by the semantics) written in that notation is also executable and can be used to infer information implied in that document. Denotational semantics of a language consists of three components: syntax, semantic algebras, and valuation functions. All three components can be specified in Horn logic. The interesting aspect about using Horn logic for specifying denotational semantics is that both the syntax as well as the semantic specification is executable. The syntax specification validates a document, while the semantic specification maps it to its executable mathematical semantics (called its denotation). The denotation of the document can then be employed for inferring implied information (i.e., querying the document). Since both the syntax and semantics are denotationally specified in Horn logic, they are declarative, and thus can be developed very rapidly.

As a result, as the mark-up language or a resource description notation rapidly evolves, its executable semantics can be developed with the same rapid pace. Thus, our framework can be used for computing the executable semantics of XML [19], RDF [25], as well as DAML [1] and OWL [15]. Providing automatic means of translating RDF descriptions into notations that are executable, not only specifies the meaning of the descriptions, but also produces a representation of the descriptions from which inferences can automatically be made using traditional automatic theorem provers, problem solvers, and other inference engines.

In the rest of this paper, we expound on our approach to semantic processing of the semantic web. For illustration purposes we use RDF and show how RDF documents can be rapidly translated into Horn logic (pure Prolog) using the Horn logical denotational method. This Prolog code can be queried to infer (or compute) information that is implied by the document. In a similar vain we show that query languages designed for resource description formats (e.g., RDQL for RDF) can be translated into Horn logic using a Horn logical denotational approach. A RDQL query is thus translated into a Prolog query, and if the semantics of RDF and RDQL are developed in a consistent manner, than the Prolog query obtained from RDQL can be executed on Prolog data obtained from the RDF document.

Horn logic has played an important role in the Semantic Web enterprise and has been used in the past to develop systems for querying RDF documents [34, 16]. The novel contribution of this paper is to show that a semantics based approach that relies on Horn Logic is sufficient to automatically (and rapidly) obtain such systems.

## 2 A Semantics Based Approach to Translation

If more information is to be inferred from a document written using a mark-up language, then it has to be semantically mapped into an (executable) notation which is amenable to drawing inferences, e.g., via querying. That is, the document has to be syntactically transformed into an executable notation in a manner that preserves its semantics. A relatively easy approach to accomplish this is to express the formal semantics of the mark-up language in terms of this executable notation. Such an approach has been developed in [23] using Horn Logic and denotational semantics. In this approach, to translate one notation, say $\mathcal{L}_s$ to $\mathcal{L}_t$, the denotational semantics of $\mathcal{L}_s$ is given in terms of the language constructs of $\mathcal{L}_t$. All components of this semantics are coded in Horn logic rendering it executable. Given a program/document coded in $\mathcal{L}_s$, its denotation under this semantics is the program/document coded in $\mathcal{L}_t$. The executable semantics thus acts as a translator. Because this semantics is denotational (declarative), it can be specified quickly. Additionally, since the translator is obtained automatically from semantics, it is provably correct w.r.t. the specification.

Thus, assuming that we have RDF as the mark-up language, the semantics of RDF can be given denotationally in terms of an executable notation such as Horn Logic (pure Prolog). Likewise, given a query language such as RDQL, its semantics can also be given denotationally in terms of Horn Logic. This Prolog translated query can be executed w.r.t. the Prolog translated RDF document to compute the query results.

### 2.1 Denotational Semantics

*Denotational Semantics* [32] is a well-established methodology for the design, description, and analysis of programming languages. In the denotational semantics approach, the semantics of a programming language/notation is specified in terms of mathematical entities, such as sets and functions. The denotational semantics of a language $\mathcal{L}$ has three components: (i) *syntax specification:* maps sentences of $\mathcal{L}$ to parse trees; it is commonly specified as a grammar in the BNF format; (ii) *semantic algebra:* represents the mathematical objects used for expressing the meaning of a program written in the language $\mathcal{L}$; these mathematical objects typically are sets or domains (partially ordered sets, lattices, etc.) along with associated operations to manipulate the elements of the sets; (iii) *valuation functions:* these are functions mapping parse trees to elements of the semantic algebras.

Given the denotational semantics of a language $\mathcal{L}$, and a program $\mathcal{P}_\mathcal{L}$ written in $\mathcal{L}$, the denotation of $\mathcal{P}_\mathcal{L}$ w.r.t. the denotational semantics can be obtained by applying the top-level valuation function to the parse tree of $\mathcal{P}_\mathcal{L}$. The denotation of $\mathcal{P}_\mathcal{L}$ is an entity that is amenable to formal mathematical processing, and thus has a number of applications. For example, it can be used to prove properties of $\mathcal{P}_\mathcal{L}$, or it can be transformed to obtain other representations of $\mathcal{P}_\mathcal{L}$ (e.g., a compiled representation that can be executed more efficiently [20]). In this

paper we assume that the reader is familiar with formal semantics. A detailed exposition can be found in [32].

### 2.2 Horn Logical Semantics

Traditional denotational definitions express syntax as BNF grammars, and the semantic algebras and valuation functions using $\lambda$-calculus. The Horn Logical Semantics of a language uses Horn-clause logic (or pure Prolog) to code all three components of the denotational semantics of a language [20]. This simple change in the notation for expressing denotational semantics, while resulting in loss of some declarative purity, leads to a number of applications [20]. There are two major advantages [20]. First, a parser can be obtained from the syntax specification with negligible effort: the BNF specification of a language $\mathcal{L}$ can be trivially translated to a *Definite Clause Grammar* (DCG) [35,31]. This syntax specification, coded as a DCG, can be loaded in a Prolog system, and a parser automatically obtained. This parser can be used to parse programs written in $\mathcal{L}$ and obtain their parse trees. The semantic algebra and valuation functions can also be coded in Horn clause logic. Second, Horn logical semantics can be used for automatic verification [20].

Since both the syntax and semantics of the specification are expressed as logic programs, they are both executable. These syntax and semantic specifications can be loaded in a logic programming system and "executed," given a program written in $\mathcal{L}$. This provides us with an interpreter that computes the semantics of programs written in the language $\mathcal{L}$.

### 2.3 Semantics-Based Language Translation

Horn logical semantics also provides a formal basis for *language translation*. Specification of a translator can be seen as an exercise in semantics. Essentially, the meaning or semantics of the language $\mathcal{L}_s$ can be given in terms of the constructs of the language $\mathcal{L}_t$. This meaning consists of both syntax and semantic specifications. If these syntax and semantic specifications are executable, then the specification itself acts as a translation system, providing a provably correct translator. The task of specifying the filter from $\mathcal{L}_s$ to $\mathcal{L}_t$ consists of specifying the definite clause grammar (DCG) for $\mathcal{L}_s$ and $\mathcal{L}_t$ and the appropriate valuation predicates which essentially relate (map) parse tree patterns of $\mathcal{L}_s$ to parse tree patterns of $\mathcal{L}_t$. Let $\mathcal{P}_s(S_s, T_s)$ be the top level predicate for the DCG of $\mathcal{L}_s$ that takes a sentence $S_s$ of $\mathcal{L}_s$, parses it and produces the parse tree $T_s$ for it. Let $\mathcal{P}_t(S_t, T_t)$ be the top level predicate for the DCG of $\mathcal{L}_t$ that takes a sentence $S_t$ of $\mathcal{L}_t$, parses it and produces the parse tree $T_t$ for it. Let $\mathcal{M}_{st}(T_s, T_t)$ be the top level valuation predicate that relates parse trees of $\mathcal{L}_s$ and $\mathcal{L}_t$. Then the relation

$$\texttt{translate}(S_s, S_t) \texttt{ :- } \mathcal{P}_s(S_s, T_s),\ \mathcal{M}_{st}(T_s, T_t),\ \mathcal{P}_t(S_t, T_t).$$

declaratively specifies the equivalence of the source and target sentence under the semantic mapping given. The `translate` predicate can be used for obtaining $S_t$ given $S_s$ (and *vice versa*).

Note that because the semantics is specified denotationally, the syntax and semantics specification of a language are guided by its BNF grammar: there is one syntax rule in the syntax specification per BNF production and one semantics rule in the semantics specification per BNF production. Occasionally, however, some auxiliary predicates may be needed while defining valuation predicates for semantics. Also, if a Prolog interpreter is to be used, extra syntax rules may be introduced if the BNF grammar has left recursive rules which need to be eliminated in the DCG (by converting to right-recursive rules).

## 3 Semantic Processing of the Semantic Web

In the context of the web, our semantics based approach to translation has a number of applications. First, to provide interoperability on the web, translators can be built that translate one format into another. Indeed this approach has been applied to translate one XML to another [23,21], as well as to convert HTML to VoiceXML in an attempt to make the web accessible via audio [11]. This helps in increasing the usability of the regular (i.e., non-semantic) web. Second, syntactic mark-up can be translated into an executable notation which can be queried to infer more knowledge from documents. This permits sophisticated applications to be deployed on the semantic web.

In this paper we explore the applications of our technique to the semantic web enterprise. We considered two languages: (i) RDF, that has been used for marking up semantic content on the web, and (ii) RDQL, that has been used for querying the semantic web. Both are denotationally translated into Horn logic using the approach outlined, to aid in processing queries to the semantic web.

Let's first consider specifying the semantics of RDF. The ("triples") statements of the RDF data model can be thought of as the equivalent of ground facts in a logic based language. Consequently, logic based approaches to information management and querying map very simply onto the RDF model. We have developed a complete denotational semantics of RDF, which consists of syntax specification and semantic specification.

### 3.1 Syntax Specification

We specify the syntax of RDF as a definite clause grammar. As stated earlier, DCG rules for RDF can be easily obtained from its BNF specification. The syntax specification in the DCG format automatically yields a parser. The formal grammar for RDF described in [25] was used to obtain the DCG. The following examples illustrate some rules from the formal grammar of RDF and their corresponding DCG rules. Note that the formal grammar in [25] is given in the Extended BNF form, while the DCG corresponds to a BNF.

```
EBNF Grammar rule:
 description ::= '<rdf:Description' idAboutAttr? bagIdAttr? propAttr* '/>'
                  | typedNode
```

```
DCG rules:
 description(descr(IAA,BIA,PA)) --> ['<rdf:Description'], idaboutattr(IAA),
                                     bagidattr(BIA), propattrs(PA), ['/>'].
 description(descr(IAA,BIA)) -->
          ['<rdf:Description'], idaboutattr(IAA), bagidattr(BIA), ['/>'].
 description(descr(IAA,PA)) -->
          ['<rdf:Description'], idaboutattr(IAA), propattrs(PA), ['/>'].
 description(descr(IAA)) --> ['<rdf:Description'], idaboutattr(IAA),['/>'].
 description(descr(BIA,PA)) -->
          ['<rdf:Description'], bagidattr(BIA), propattrs(PA), ['/>'].
 description(descr(BIA)) --> ['<rdf:Description'], bagidattr(BIA), ['/>'].
 description(descr(PA)) --> ['<rdf:Description'], propattrs(PA), ['/>'].
 description(descr([])) --> ['<rdf:Description'], ['/>'].
 description(descr(TN)) --> typenode(TN).
 propattrs(PA)-->propattr(PA).
 propattrs((PA,PAs))-->propattr(PA), propattrs(PAs).
```

The above EBNF grammar rule states that a description consists of the 'rdf:Description' tag followed by an optional id describing what the document is about, followed by an optional id for the associated bag, followed by zero or more property attributes, and terminating with the symbol '/>'. The DCG rules corresponding to this EBNF rule is also shown above. This DCG is slightly extended, so that the parse tree of the document is recursively synthesized during the parsing process. The parse tree of the document generated is passed to the semantic valuation predicates, for mapping to its denotation.

Another example EBNF rule and its correspond DCG rules are shown below.

```
EBNF Grammar rule:
    propAttr ::= typeAttr | propName '="' string '"'

DCG rules:
    propattr(propattr(TA))-->typeattr(TA).
    propattr(propattr(PN,S))-->propname(PN),['="'],string(S),['"'].
```

The following example shows an input RDF statement and the corresponding output that is obtained from the DCG,

```
RDF statement:
   <rdf:RDF>
        <rdf:Description about="http://www.w3.org">
            <s:Publisher>World Wide Web Consortium</s:Publisher>
            <s:Title>W3C Home Page</s:Title>
            <s:Date>1998-10-03T02:27</s:Date>
        </rdf:Description>
   </rdf:RDF>

Output from DCG (after parsing):
  R = r(o(descr(idaboutattr(aboutattr(
                uriref(string("http://www.w3.org")))),
```

```
        (propertyelt(propname(qname(nsprefix(s), name(publisher))),
               value(string("World Wide Web Consortium)),
               propname(qname(nsprefix(s), name(publisher)))),
         propertyelt(propname(qname(nsprefix(s), name(title))),
               value(string(W3C Home Page)),
               propname(qname(nsprefix(s), name(publisher)))),
         propertyelt(propname(qname(nsprefix(s), name(date))),
               value(string(1998-10-03T02:27)),
               propname(qname(nsprefix(s), name(date)))
    ))) ) )
```

### 3.2 Semantic Specification

RDF statements can be represented as logical expressions. However, instead of defining a triple `(P, S, O)` directly we adopt the approach of [26] and define it as a ternary relation "property": `property(S, P, O)`.
which means resource `S` has property `P` with value `O`. The reason is that developing statements about properties would lead to unwanted higher order statements. Our goal is to keep the semantics first order, not simply because we are interested in using a logic programming based framework, but also because higher order logics quickly become intractable and inefficient. Therefore, we represent RDF document as a set of assertions having three arguments. These assertions are generated during semantic processing, which uses the parse tree produced from the syntax specification as input. The output parse tree obtained is semantically processed to identify the subject, predicate and object, which in turn, are used to generate the ground facts. The semantic function is specified denotationally in Horn Logic. The semantic function (actually, a predicate) maps the parse tree to a database of ground facts.

The following example illustrate fragments of rules used to map property elements to a list of ground facts. The rules essentially map the corresponding parse tree obtained in the semantic phase to the subject, predicate and object values in order to generate the Prolog fact database. The property elements are processed recursively to obtain the list of facts.

```
description(descr(IAA,PE),IAASTR,PROSTR,VALSTR) :-
          idaboutattr(IAA,IAASTR), propertyelts(PE,PROSTR,VALSTR,IAASTR).

propertyelts((PE,PEs),PROSTR,VALSTR,IAASTR) :-
         propertyelt(PE,IAASTR,PROSTR,VALSTR),
         assemble(IAASTR,PROSTR,VALSTR)
         propertyelts(PEs,PROSTR1,VALSTR1,IAASTR).
```

In the first rule, the value of `IAASTR`, which corresponds to the "subject" argument of the property predicate, is obtained from the parse tree. The value of `IAASTR` is then used to retrieve the values of the "predicate" argument and the "object" argument of the property predicate to `assemble` the related facts. We can regard the predicate `property` as an operation in the semantic algebra.

The following examples illustrate some RDF statements and their corresponding denotation generated as ground facts by our semantic specification.

```
RDF statement 1:
   <rdf:RDF>
       <rdf:Description about="http://www.w3.org/Home/Lassila">
           <s:Creator>
                <rdf:Description about="http://www/w3.org/staffID/85740">
                   <v:Name>Ora Lassila</v:Name>
                   <v:Email>lassila@w3.org</v:Email>
                 </rdf:Description>
            </s:Creator>
       </rdf:Description>
   </rdf:RDF>

DENOTATION of statement 1:
   property("http://www.w3.org/Home/Lassila", creator,
                             "http://www.w3.org/staffID/85740").
   property("http://www.w3.org/staffID/85740", name, ora lassila).
   property("http://www.w3.org/staffID/85740", email, lassila@w3.org).
```

The RDF model also defines the concept of containers. Frequently, it is necessary to refer to a collection of resources; for example, to say that a work was created by more than one person, or to list the students in a course. RDF containers are used to hold such lists of resources or literals. A container object can be a *bag*, a *sequence* or an *alternative*. To semantically map the container tag, we make use of structures which holds such list of resources or literals. The name of the structure identifies the type of the container (a bag, a sequence or an alternative). The first argument of the structure is reserved for the ID of the container. If there is no ID for the container then this argument takes the default value of `noid`. The second argument of the structure is a list of resources or literals contained within the container. The following example illustrates the bag container.

Consider an RDF rendition of the information contained in the following sentence: "The students in course 6.001 are Amy, Tim, John and Mary." The corresponding RDF document and its denotation generated by our semantics is shown below.

```
RDF statement 2:
    <rdf:RDF>
       <rdf:Description about="http://mycollege.edu/courses/6.001">
          <s:students>
             <rdf:Bag>
                <rdf:li resource="http://mycollege.edu/students/Amy"/>
                <rdf:li resource="http://mycollege.edu/students/Tim"/>
                <rdf:li resource="http://mycollege.edu/students/Mary"/>
             </rdf:Bag>
          </s:students>
       </rdf:Description>
    </rdf:RDF>
```

```
DENOTATION of statement 2:
   property("http://mycollege.edu.courses/6.001", students,
             bag("StudentsOf6.001", ["http://mycollege.edu/students/Amy",
                           "http://mycollege.edu/students/Tim",
                           "http://mycollege.edu/students/Mary")))
```

In the above example the ID of the Bag container is "StudentsOf6.001" and the Bag contains three resources.

In addition to making statement about web resources, RDF can also be used for making statements about other RDF statements. These are referred to as *higher order statements*. In order to express higher order statements RDF defines a model of a statement. This model of a statement has four properties, namely, subject, predicate, object and type, and is also known as a reified statement. The semantics of reified statements is defined with the help of a a new predicate called `statement`. By assigning internal ids, higher order statements can be represented in Horn logic. To achieve this, the first argument of the "statement" is bound to a unique id which links this reified statement to its other properties. The second argument of "statement" is the subject of the reified statement, the third argument is the predicate of the reified statement and the fourth argument is the object of the reified statement. Other additional statements made about this reified statement use the unique id to refer to the reified statement. The following example shows the semantic mapping of a higher order statement.

Consider the statement: "Ralph Swick says that Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila." The RDF document generated for this statement and the corresponding denotation generated by our semantics is shown below. Note that the two predicates in the denotation are joined via the unique id `id007`.

```
RDF statement 3:
 <rdf:RDF>
   <rdf:Description>
     <rdf:subject resource="http://www.w3.org/Home/Lassila"/>
     <rdf:predicate resource="http://description.org/schema/Creator"/>
     <rdf:object>Ora Lassila</rdf:object>
     <rdf:type resource="www.w3.org/1999/02/22-rdf-syntax-ns#Statement"/>
     <a:attributedTo>Ralph Swick</a:attributedTo>
   </rdf:Description>
 </rdf:RDF>


DENOTATION of statement 3:
   statement(id007, "http://www.w3.org/Home/Lassila",
                "http://description.org/schema/Creator", Ora Lassila)
   property(id007, attributedTo, Ralph Swick)
```

The semantics of other RDF constructs are similarly defined. Name-spaces are also easily handled by associating name-space identifiers with their corresponding URI via a predicate. Details are omitted due to lack of space. The

whole semantic specification for RDF has indeed been developed. The semantic specification coupled with syntax specification can produce the denotation of any arbitrary RDF document as a set of logic programming ground facts.

### 3.3 Syntax and Semantics of RDQL

Several RDF Query languages and systems have been recently proposed. These proposals either take the "database approach" or the "knowledge base approach" to query language design (see [9]). The database approach interprets an RDF document primarily as an XML document, which leads to several drawbacks described in [9]. For this reason, all of the recently presented RDF Query languages and systems adhere to the knowledge base approach, explicitly dealing with RDF triples instead of document trees. It is easy to see that the knowledge base approach of RDF querying fits our approach quite well: queries can be expressed as (Horn logic) rules, which can be processed with respect to the denotation of the RDF document using a Logic Programming engine.

The semantics of RDF Query languages like RDQL [9], RQL [3], RDFDBQL [30] or SQUISH [12] can be expressed in Horn logic, using the approach described above. We choose RDQL as an example RDF Query language. RDQL(RDF Data Query Language) is an implementation of SQL-like query language for RDF. It treats RDF as data and provides query with triple patterns and constraints over a single RDF model. An RDQL query is of the form:

```
SELECT vars
FROM documents
WHERE expressions
AND filters
USING Namespace declarations
```

where `SELECT` clause indicates which RDQL variables should be returned by the query; the `FROM` clause of the query indicates the RDF sources to be queried (each source is enclosed by angle brackets(<&>)); while the `WHERE` clause indicates the constraints that RDF triples (subject, predicate, object) that constitute the solution must satisfy. The `WHERE` part is expressed by a list of restrictions separated by commas, each restriction takes the form (subject, predicate, object) where the subject, predicate and object can be a literal value or a RDQL variable. Finally, the `USING` clause declares all the name spaces that will be used for RDF properties.

RDQL queries are also converted to Horn Logic queries, using the same method as described for RDF statements. A syntax specification for RDQL as a DCG is developed based on the RDQL formal grammar described in [9]. The denotational semantics of RDQL as a mapping from parse trees to logic programming facts and queries is also specified. Given a RDQL query, its denotation can be viewed as a query coded in logic programming. The logic programming coded query can be executed on top of the logic programming coded database obtained denotationally from the RDF document. The following example shows a sample RDQL query and its corresponding denotation that is generated:

```
RDQL Query:                          DENOTATION:
    SELECT ?a ?b                       :- property(X, pred, Y), Y > 250.
    WHERE (?a, pred, ?b)
    AND    ?b > 250
```

### 3.4 An Example

The following example shows some RDF statements and RDQL queries. The Prolog denotation corresponding to these statements and queries is also shown.

```
RDF statement:
   <rdf:RDF>
        <rdf:Description about="http://www.abc.org">
             <s:Author>John</s:Author>
             <s:Title>ABC Home Page</s:Title>
             <s:Hits>200</s:Hits>
        </rdf:Description>
        <rdf:Description about="http://www.pqr.org">
             <s:Author>Tom</s:Author>
             <s:Title>PQR Home Page</s:Title>
             <s:Hits>350</s:Hits>
        </rdf:Description>
   </rdf:RDF>

RDF DENOTATION:
    property("http://www.abc.org", author, john).
    property("http://www.abc.org", title, abc home page).
    property("http://www.abc.org", hits, 200).
    property("http://www.pqr.org", author, Tom).
    property("http://www.pqr.org", title, pqr home page).
    property("http://www.pqr.org", hits, 350).

Query in RDQL (constraints):
    SELECT ?a ?b
    WHERE (?a, hits, ?b)
    AND    ?b > 250

RDQL DENOTATION:
    :- property(X, hits, Y), Y > 250.
```

Once the denotation of RDF and RDQL is obtained in terms of logic programming, the RDQL query can be executed on top of the data provided by RDF using a logic programming engine. For example, execution of the query

```
:- property(X, hits, Y), Y > 250.
```

will yield the answer

```
X = "http://www.pqr.org", Y = 350.
```

## 4 An Extended Example

In this section we give a slightly extended example that illustrates how the techniques discussed in this paper can be applied to build an on-line application that integrates, combines and deduces information that is on the semantic web to assist users in performing tasks. These examples are motivated from the *semantic web challenge* [6]. While a true semantic web is still far, our research efforts reported here, we believe, will bring us one step closer.

Consider the case of disaster management. Given the Semantic Web, we should be able to use and integrate data from various sources to provide actual and accurate support in a disaster situation (e.g., the flood in Germany in August 2002 or the forest fires in the USA). Possible information sources in this case are (i) weather information, (ii) maps from geographical information systems (iii) News sites and (iv) satellite images. We assume that this information is available as RDF documents, which are denotationally mapped to a logic program using our technique. Thus, from the weather information sites we can obtain information as shown below (instead of showing the RDF, we give an English description). We kept the level of treatment very simple, in reality, the RDF document and the information generated may be more complex.

*Today it is rainy in plano:*
`property(plano, weathertoday, rainy).`
*The weather in plano will be cold tomorrow:*
`property(plano, weathertomorrow, cold).`
*Plano will have sunny weather next week:*
`property(plano, weathernextweek, sunny).`

Similarly, from the transport information RDF we can obtain the following data:

| | |
|---|---|
| *Frisco can be reached by bus#1:* | `property(frisco, access, bus#1).` |
| *Plano can be reached by bus#2;* | `property(plano, access, bus#2).` |
| *Plano can be reached by bus#3;* | `property(plano, access, bus#3).` |

Information regarding routes can be obtained from the map sites. The following predicates are generated from the RDF documents of the Map sites:

| | |
|---|---|
| *Bus#1 runs on route 75N:* | `property(bus#1, route, 75N).` |
| *Bus#2 runs on route 635-North:* | `property(bus#2, route, 635-North).` |
| *Bus#3 runs on plano road:* | `property(bus#3, route, plano-road).` |

Similarly from the news sites we can obtain the following information:

*Route 635-North is closed for construction:*
`property(635-North, status, nonoperational).`
*Bus #3 is not operational today:*
`property(bus#3, status, nonoperational).`

From the Medical information sites we can obtain the following data:

| | |
|---|---|
| *Hospital#1 is located in plano.* | `property(hospital#1, location, plano).` |
| *Hospital#2 is located in frisco.* | `property(hospital#2, location, frisco).` |
| `20 beds are free in Hospital#1.` | `property(hospital#1, freebeds, 20).` |
| `30 beds are free in Hospital#2.` | `property(hospital#2, freebeds, 30).` |

With the above data we can provide answers to various different queries. These queries, posed in RDQL, are denotationally translated into Horn logic. The English equivalent of the RDQL queries, and their corresponding denotations are given below.

*What weather is predicted in plano tomorrow?:*

```
:- property(plano, weathertomorrow, X).
```

*By which bus can one reach frisco today?:*

```
:- property(frisco, access, X),
   property(X, route, Y),
   property(Y, status, Z), Z ≠ nonoperational.
```

*Which hospitals in plano have free beds?:*

```
:- property(X, location, plano),
   property(X, freebeds, Y), Y > 0.
```

*Are bus services to Plano operational today?:*

```
:- property(plano, access, B), property(B, route, T),
   property(T, status, X), X ≠ nonoperational.
```

While these queries are relatively simple, more complex queries can be asked, and as long as they are expressible in RDQL, they can be processed using our approach.

## 5 Discussion

The approach outlined in this paper for turning a syntactically marked-up document into a semantically understandable, executable form is quite universal. In fact, it has been applied to quite diverse scenarios [23]. For example, it has been used to map complex, multiple notations used by bioinformatics systems to each other [23] so that they are inter-operable; it has been used to systematically translate relational databases (data as well as queries) to object-oriented databases [14], as well as to translate Braille based mark-up notation (called Nemeth Math notation [10]) for mathematics to LaTeX. In the context of the Web, this approach has been used to automatically generate validating parsers from XML DTDs (the semantics of the language used to describe DTDs is expressed in terms of a DCG [22]; the generated DCG acts as a validating parser for the input XML DTD) and for translating HTML documents to VoiceXML in

order to make the Web accessible via audio [11]. Currently, work is in progress to use our semantics based methodology for interfacing intelligent multi-agent systems with other agents through DAML [1].

In the context of querying, this approach has been applied to reason about properties of real-time systems [24] where a real-time system specified as a timed-automata is denotationally mapped to a constraint logic program over reals [27]. The generated CLP(R) program is queried to reason about timing properties. A similar approach has been used to verify properties of the Bay Area Rapid Transit system specification; in this case, the design of the BART system specified in Ada is semantically mapped to a logic program, which is then queried to prove *safety* and *liveness* properties [28].

Our approach to querying RDF documents illustrates the power of logic, since not only the target executable notation is based on logic, the means of semantically translating RDF into logic also relies on logic. The importance of logic in realizing the semantic web is well-realized by the community, our paper provides yet another venue where logic can help in piecing together a true semantic-web. The main advantage of our approach is that the semantic translators can be rapidly developed. In fact, it took us only a few weeks of work to specify the syntax and semantics of both RDF and RDQL. As a result, as mark-up languages for the web rapidly evolve, as they have from HTML to XML to DAML to RDF, they can be processed and queried with the same rapidity.

## 6 Related Work

RDF provides a good framework for effectively storing and retrieving data, especially if multiple heterogeneous vocabularies are involved. Several concepts have been proposed so far concentrating on different goals (e.g. expressiveness, performance). Our concept follows the approach of applying Logic Programming as it provides the expressivity that we need to build generic transformation services.

A number of tool already exist that exploit the relationship between the W3C's RDF and Logic Programming (see [17]). Such efforts include TRIPLE [34], a layered rule language that aims to support applications in need of RDF reasoning and transformation. The core language is based on Horn logic which is syntactically extended to support RDF primitives like name-spaces, resources and statements. Metalog [29] is another query/logical-reasoning system for the semantic web. Metalog views an RDF statement in the logical setting as just a binary predicate involving the subject and the literal. The Metalog schema extends plain RDF with a "logical layer," enabling expression of arbitrary logical relationships within RDF. SiLRI (Simple Logic Based RDF Interpreter) [16] is a main-memory logic-based inference engine implemented in JAVA. It implements a major part of Frame Logic and has support for RDF. The implemented semantics include well-founded semantics and the rule language allows general logic programs. Various other RDF Query languages like RQL [3], R-DEVICE [4], RDFQL [5], Versa [7], N3 [8], RDQL [9], have also been developed and used.

All these approaches require considerable implementation effort, making it hard for them to keep up with the pace of evolution of the semantic web. In contrast, our approach has firm semantic foundations and allows for rapid mapping to executable denotations that can be used for inferring information implicit in Semantic Web documents.

## 7 Conclusions

In this paper we presented a semantics-based approach to mapping mark-up languages to executable notations so that documents coded in these mark-up languages can be queried further. Our approach is based on denotational semantics and Horn logic. In the denotational approach, the semantics of a language is expressed in the mathematical formalism of the $\lambda$-calculus. We adopted an approach in which Horn logic instead of the $\lambda$-calculus is the formalism used to express the semantics. We demonstrated our approach by developing the Horn logical denotational semantics for RDF and RDQL, and showing how it leads to their becoming executable. The main advantage of our approach is that documents can be rapidly mapped to their executable semantics, which can be used for querying them. Using this approach we were able to develop a complete semantics of RDF in a few of weeks of work. A complete Horn logical executable semantics of RDF and RDQL indeed will be publicly available shortly.

## References

1. DARPA Agent Mark-up Language. `http://www.daml.org`.
2. The Semantic Web. `http://www.semanticweb.org`.
3. RQL: The RDF Query Language. `http://139.91.183.30:9090/RDF/RQL/`
4. R-DEVICE. `http://lpis.csd.auth.gr/systems/r-device.html`
5. Intellidimension. Intellidimension's RDF Gateway. See `http://www.intellidimension.com/RDFGateway`
6. The Semantic Web Challenge. `http://challenge.semanticweb.org/`.
7. Versa: An RDF Query Language. `http://rdfinference.org/versa-about.xml?xslt=index.xslt`
8. RDF:Notation3. `http://www.gingerall.com/charlie/ga/html/rdf-notation3.html`
9. RDQL: Jena toolkit – RDQL Query Language. `http://www.hpl.hp.com/semweb/rdql.html`
10. N. Annamalai, G. Gupta, et al. INSIGHT: A System for Translating Braille based Mathematical Documents to LaTeX. In *Proc. HCI 2003*. pp 1245–1249.
11. N. Annamalai. An Extensible Translator from HTML to VoiceXML. MS Thesis. May 2002. University of Texas at Dallas.
12. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame:A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proc. International Semantic Web Conference*. Springer LNCS 2342, 2002.
13. A. S. Christophides, V. Plexousakis, D. Karvounarakis, and M. Scholl. RQL: A declarative query language for RDF. In *11th Int'l World Wide Web Conference (WWW'02)*, 2002.

14. N. Datta. A Semantic based approach to Interoperation between Object Querly Language and ODBC. MS thesis. New Mexico State University. Aug. 2000.
15. M. Dean, D. Connolly, et al. OWL Web Ontology Language 1.0 Reference. W3C Working Draft, Mar 2003. http://www.w3.org/TR/owl-ref/.
16. S. Decker, D. Brickley, J. Sarrela, J. Angele. SiLRI (Simple Logic Based RDF Interpreter). http://www.aifb.uni-karlsruhe.de/~sde/rdf/.
17. S. Decker. Logic Databases on the Semantic Web: Challenges and Opportunities. In *Proc. Int'l Conf. on Logic Programming*. Springer LNCS. 2002. pp. 20–21
18. R. V. Guha Rdfdb : An RDF database. http://rdfdb.sourceforge.net. 2000.
19. C. Goldfarb, P. Prescod. The XML Handbook. Prentice Hall. 1998.
20. G. Gupta "Horn Logic Denotations and Their Applications," *The Logic Programming Paradigm: A 25 year perspective* Springer Verlag. pp. 127–160.
21. G. Gupta, Building the Tower of Babel: Interconverting XMLs for Interoperability. Proc. In *7th Int'l Conf. on Computers Helping People with Special Needs (ICCHP00)*. pp. 267–272.
22. G. Gupta, X. Zhou. Automatic Generation of Validating Parsers for XML. Internal Report. U.T. Dallas. 2001.
23. G. Gupta, H-F. Guo, A. Karshmer, E. Pontelli, et al. Semantic-Based Filtering: Logic Programming's Killer App? In *4th Int'l Symp. on Practical Aspects of Declarative Languages*, Springer LNCS 2257, pp. 82–100, 2002.
24. G. Gupta, E. Pontelli. "A Constraint-based Approach to Specification and Verification of Real-time Systems," In *Proc. Real-time Symposium*, IEEE Press. pp. 230–239. 1997.
25. P. Hayes (editor). RDF Model Theory. W3C Working Draft http://www.w3.org/TR/rdf-mt/
26. G. Karvounarakis RDF Query languages: A state-of-the-art, 1998. http://139.91.183.30:9090/RDF/publications/state.html.
27. J. L. Lassez and J. Jaffar. Constraint logic programming. In *Proc. 14th ACM POPL*, 1987.
28. L. King, G. Gupta, E. Pontelli. Verification of a Controller for BART: An Approach based on Horn Logic and Denotational Semantics. In *High Integrity Software Systems*. Kluwer Academic Publishers.
29. Massimo Marchiori, Janne Saarela. Query + Metadata + Logic = Metalog, http://www.w3.org/TandS/QL/QL98/pp/metalog.html
30. L. Miller RDF Query : Squish QL, 2001. http://swordfish.rdfweb.org/rdfquery/
31. R. A. O'Keefe. The Craft of Prolog. MIT Press. 1990.
32. D. Schmidt. *Denotational Semantics: a Methodology for Language Development*. W.C. Brown Publishers, 1986.
33. D. Schmidt. Programming language semantics. In CRC Handbook of Computer Science, Allen Tucker, ed., CRC Press, Boca Raton, FL, 1996. Summary version, ACM Computing Surveys 28–1 (1996) 265–267.
34. M. Sintek, S. Decker. TRIPLE: An RDF Query, Inference and Transformation Language. http://triple.semanticweb.org.
35. L. Sterling & S. Shapiro. The Art of Prolog. MIT Press, '94.