HOW TO QUERY THE GENEONTOLOGY

Andreas Doms[1], *Tim Furche*[2], *Albert Burger*[3] *and Michael Schroeder*[1]

[1]Biotec, Technical University Dresden, Germany
[2]Ludwig-Maximilians-Universität Munich, Germany
[3]Medical Research Council, Human Genetics Unit, Edinburgh, UK

ABSTRACT

Ontologies, which are structured, hierarchical vocabularies, are widely used in molecular biology to annotate sequence and structure data. One such ontology, the GeneOntology, contains some 18000 terms on biological processes, molecular function, and cellular components. GeneOntology is available as flat file, in web formats such as XML and RDF, and as database. Using these formats we compare three different reasoners to query the GeneOntology. Prolog is the classical logic programming approach to reason over the ontology, Prova is a rule-based Java scripting language, and Xcerpt a query language for XML and RDF. We conclude by discussing the strengths and weaknesses of the three approaches.

1. INTRODUCTION

Recently much work has been dedicated to create ontologies and annotate other data sources with the terms of the ontologies. One prominent example is the GeneOntology, which defines a vocabulary for cellular components, biological processes and molecular function. When searching the data sources, which have been annotated with ontological terms reasoning over the ontology becomes crucial. A query for all genes, which have been annotated with "small GTPase mediated signal transduction" should also return all genes, which have been annotated with the children of the term, such as Ras, Rho, or Rac mediated signal transduction. I.e. a reasoner is required, which recursively traverses the is-a hierarchy to find associations. Such a reasoner needs to be able to handle boolean expressions correctly. E.g. which genes have been annotated with small GTPases mediated signal transduction apart from Rho. The GeneOntology is available in various formats such as flat files, Web formats such as XML and RDF, and as database. We compare the three different reasoners Prolog, Prova, and Xcerpt to query the GeneOntology.

2. ONTOLOGIES IN BIOINFORMATICS

Currently there is no agreed vocabulary used in molecular biology. For example, gene names are not used in a consistent way. GeneCards ([13]) and LocusLink ([12]) address this problem by providing aliases. For example GeneCards lists six aliases for a gene that is responsible for breast cancer

- *PSCP*,
- *RNF53*,
- *breast cancer 1, early onset*,
- *breast-ovarian cancer, included*,
- *papillary serous carcinoma of the peritoneum* and
- *Breast cancer type 1 susceptibility protein*.

At the time of writing, searching PubMed for *PSCP* returns 2035 relevant articles. Searching for *papillary serous carcinoma of the peritoneum*, returns 78 articles. However, searching for both terms returns only 15 hits. In general, there is a pressing need in molecular biology to use common vocabularies. This need has been addressed through the ongoing development of biomedical ontologies. Starting with the GeneOntology[1] [7], the Open Biomedical Ontologies effort[2] currently hosts over 50 biomedical ontologies. Fig. 1 shows some example ontologies such as *cell type*, *Drosophila development*, *Fungal gross anatomy*, *SwissProt organismal classification*, and *C. elegans development*.

2.1. Gene Ontology (GO)

A core ontology is the GeneOntology, which contains over 18000 terms describing biological processes, molecular functions, and cellular components for gene product. The biological process ontology deals with biological objectives to which the gene or gene product contribute. A process is accomplished via one or more ordered assemblies of molecular functions. The molecular function ontology deals with the biochemical activities of a gene product. It describes what is done without specifying where or when the event takes place. The cellular component ontology describes the places where a gene product can be active. The GO ontologies have become a de facto standard and is used by many databases as annotation vocabulary.

There are a number of tools that can be used to browse and query the GeneOntology. The GO website lists tools such as AmiGO and DAG-Edit, which allows users to search and browse by term or gene name. The association of ontology terms and genes is qualified by evidence

[1]www.geneontology.org
[2]obo.sourceforge.net

⊟ anatomy
  ⊞ gross anatomy
  ⊡ cell type
  ⊡ BRENDA tissue / enzyme source
⊞ chemical
  ⊡ chemical entities of biological interest
  ⊡ physico-chemical methods and properties
  ⊡ physico-chemical process
⊞ development timeline
  ⊞ animal development
  ⊞ plant development
⊞ ethology
  ⊡ Habronattus courtship
  ⊡ Loggerhead nesting
⊞ experimental conditions
  ⊡ biological imaging methods
  ⊡ microarray experimental conditions
  ⊡ physical-chemical methods and properties
  ⊡ plant enviromental conditions
⊞ genomic and proteomic
  ⊞ gene product
  ⊡ gene structure and variation
⊞ phenotype
⊞ taxonomic classification

Figure 1. Example categories of ontologies listed at the Open Biomedical Ontologies website

codes, which indicate the support for the link such as traceable author statements, statements inferred from expression pattern, from electronic annotation, or a curator. The GeneOntology is available in various formats such as flat files, the extensible mark-up language (XML), the resource description format (RDF), and as MySQL database. Fig. 2 shows an excerpt of the GO in XML. The GeneOntology can be queried using all of the above representations, which have different advantages and disadvantages. Before discuss these in more detail, we will introduce an example scenario in signal transduction.

3. EXAMPLE PROBLEMS: SMALL GTPASES

Consider Fig. 3, which shows the biological process of small GTPase mediated signal transduction with its children such as Rac, Ras, and Rho protein signal transduction. GTPases are a large family of enzymes that can bind and hydrolyze guanosine triphosphate, GTP. GTPases play an important role in signal transduction at the intracellular domain of transmembrane receptors, including recognition of taste, smell and light. GTPases also play a role in other cellular functions such as protein biosynthesis, control and differentiation during cell division, translocation of proteins through membranes and transport of vesicles within the cell. There are various subfamilies such as Ras, Rho, Rab, Arf, Ran, Rheb, Rad and Rit. Figure 4 shows a Ras protein which is in the active state bound to guanosine triphosphate, GTP.

In the context of GTPases, we will show how to answer the following three queries with three different reasoning engines:

OBO XML

```
<?xml version="1.0" encoding="UTF-8"?>
<obo>
  ...
  <term>
    <id>GO:0008150</id>
    <name>biological_process</name>
    <namespace>biological_process</namespace>
    <def>
      <defstr>A phenomenon marked by changes that ...
...lead to a
      particular result, mediated by one or more ...
...gene
      products.</defstr>
      <dbxref>
        <acc>curators</acc>
        <dbname>GO</dbname>
      </dbxref>
    </def>
    ...
    <is_root>1</is_root>
  </term>
  <term>
    <id>GO:0007275</id>
    <name>development</name>
    <namespace>biological_process</namespace>
    <def>
      <defstr>Biological processes specifically ...
...aimed at the
      progression of an organism over time from an ...
...initial
      condition (e.g. a zygote, or a young adult) ...
...to a later
      condition (e.g. a multicellular animal or an ...
...aged adult).
      </defstr>
      <dbxref>
        <acc>ems</acc>
        <dbname>WB</dbname>
      </dbxref>
    </def>
    <comment>Note that this term was 'developmental...
... process'.
    </comment>
    ...
    <is_a>GO:0008150</is_a>
  </term>
  ...
</obo>
}
```

Figure 2. Example fragment of the GeneOntology in OBO XML format

1. Show the ID of the GO term *small GTPase mediated signal transduction*!

2. Which small GTPase mediated signal transduction processes are known?

3. Which small GTPase mediated signal transduction processes apart from Rho are listed in the GO?

Much more complex queries integrating also other biomedical data sources are possible. For clarity we concentrate in this survey on the three afore-mentioned basic queries. Expressing such queries using logic-programming style reasoning-aware query languages is investigated from three perspectives: the well-established logic programming language Prolog, a rule-based scripting language, Prova, that adds access to Java objects, and Xcerpt, an XML and RDF query language based on logic programming.

▷ Biological process
  ▷ Cellular process
    ▷ cell communication
      ▷ signal transduction
        ▷ intracellular signaling cascade
          ▷ **small GTPase mediated signal transduction**
            ▷ Rac protein signal transduction ⊞
            ▷ Ras protein signal transduction ⊞
            ▷ regulation of small GTPase mediated signal transduction
              ▷ regulation of Rho protein signal transduction
                ▷ positive regulation of Rho protein signal transduction
                ▷ negative regulation of Rho protein signal transduction
            ▷ Rho protein signal transduction
              ⊐ regulation of Rho protein signal transduction
                ▷ positive regulation of Rho protein signal transduction
                ▷ negative regulation of Rho protein signal transduction
        ⊐ regulation of signal transduction
          ▷ regulation of small GTPase mediated signal transduction ⊞
    ⊐ regulation of cellular process
      ▷ regulation of signal transduction
        ▷ regulation of small GTPase mediated signal transduction
▷ Molecular function
  ▷ enzyme regulator activity
    ▷ GTPase regulator activity
      ▷ small GTPase regulatory/interacting protein activity
▷ Cellular component

Figure 3. View on the GO hierarchy containing terms related to small GTPases. ▷ symbolizes an *is_a relation and* ⊐ *symbolizes a* part_of relation. ⊞ marked nodes hide more children related to "'small GTPase mediated signal transduction"'. Note: this tree view is stripped down to the concepts of GO necessary to explain our example. The subtree related to *regulation of Rho protein signal transduction* is present two times because this GO term has multiple parents. The relations in GO are graph-shaped, we show here a simplified hierarchical representation.

4. RULES AND REASONING

Ontologies as introduced in section 2 are directed acyclic graphs and each is-a and part-of relationship can be expressed as a rule. To reason over rules there are many different formalisms and reasoning engines. One of the most well-established approaches is logic programming with programming languages such as Prolog, which allow programmers to specify problems in the form of if-then rules. Prolog is ideal if a problem can be completely specified by declarative if-then rules. Often practical consideration require however integration of non-logic programming constructs. Examples are the ability to call external web services, to exchange messages, to access underlying databases and to use Java or other programming languages. Prova ([10]) aims to close this gap and marry the benefits of rule-based programming with Java programming. Prova closely resembles Prolog syntax, but additionally provides predicates to send and receive messages, to call web services, and to access databases. Prova variables correspond to underlying Java objects, which means that all of the available Java methods are transparently accessible from within Prova. While Prolog is not suitable for programming on the web, Prova's link to Java ensures that it can work with web technologies such as XML and RDF. Xcerpt ([2]), the third rule-based language discussed in this paper, puts these web technologies at center stage. Xcerpt implements rule-based querying of XML and RDF documents. Below we give a brief introduction into Prolog, Prova and Xcerpt. Then we will show how to use them to query GeneOntology and compare their strengths and weaknesses for this task.

4.1. Prolog

Prolog (Programming in Logic) is a declarative language (vs. procedural language) which means that it describes "'what to do"' rather than "'how to do it"'. A Prolog programme consists of facts and rules. Consider e.g. the listing for the same generation problem, where :- is read as "if" and the comma as "and".

Listing 1. The same generation problem

```
1 sg(X,Y) :- parent(X,Z), parent(Y,Z).
2 sg(X,Y) :- parent(X,Z1), parent(Y,Z2), sg(Z1,Z2).
3
4 parent(bob,mary).
5 parent(bill,mary).
6 parent(bert,marge).
7 parent(marge,pete).
8 parent(mary,pete).
```

The programme specifies in just two rules under which circumstance X and Y are in the same generation. The

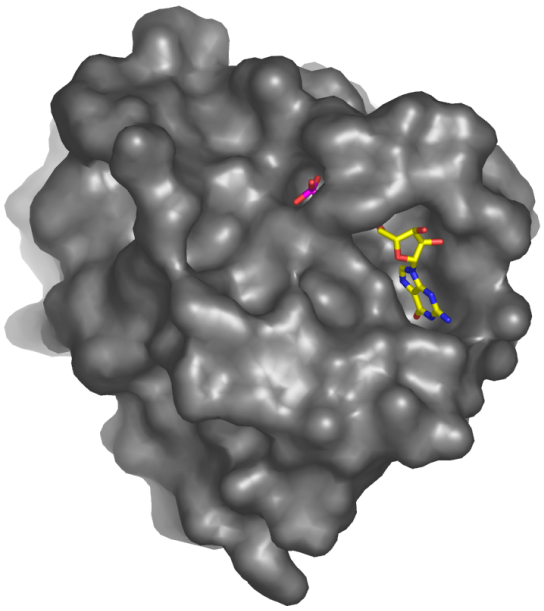

Figure 4. Crystal structure of rasa59g in the gtp-bound form

first rules states that X and Y are in the same generation if they are siblings. The second rule specifies that X and Y are in the same generation if their parents are. The query sg(bob,X) answers e.g., who is in the same generation as bob.

There are a few different implementations of Prolog, the more popular ones are: Sicstus Prolog[3], SWI-Prolog[4] and IF Prolog[5].

4.2. Prova

Prova is based itself on Mandarax[6]and provides a rule-based Java scripting language. The use of rules allows one to declaratively specify the integration needs at a high-level without any implementation details. The transparent integration of Java caters for easy access and integration of database access, web services, and many other Java services. This way Prova combines the advantages of rule-based programming and object-oriented programming in Java. This section presents rationales and design principles behind the Prova language. The Prova language is positioned as a platform for knowledge-intensive ontology-rich applications in biomedical research. It aims to satisfy the following design goals with the proposed Prova language:

- Combine the benefits of declarative and object-oriented programming;
- Merge the syntaxes of Prolog, as rule-based language, and Java as object-oriented languages;
- Expose logic as rules;
- Access data sources via wrappers written in Java or command-line shells like Perl;
- Make all Java API from available packages directly accessible from rules;
- Run within the Java runtime environment;
- Be compatible with web- and agent-based software architectures;
- Provide functionality necessary for rapid application prototyping and low cost maintenance.

Consider the following Prova code showing how knowledge can be inferred from the available facts.

Example 1 *(Declarative programming)*
Consider a table of interacting proteins. We wish to infer all interactions, direct or indirect. In Prova, this can be specified as follows (`:-` *is read as "if"):*

Listing 2. Prova example

```
% Facts (what we know)
interactDirect(a,b).
interactDirect(b,c).
interactDirect(c,d).

% Rules (how to derive new knowledge)
interact(X,Y):-interactDirect(X,Y).
interact(X,Z):-interactDirect(X,Y),interact(Y,Z).
```

The query `:- solve(interact(a,X)).`*, which can be read as "which proteins* `X` *interact with protein* `a`*?", will return the three answers* `X=b`*,* `X=c`*, and* `X=d`*.*

Thus, Prova follows classical Prolog closely by declaratively specifying relationships with facts and rules. Now let us consider two examples, where access to Java methods is directly integrated into rules.

Example 2 *(Object-oriented programming)*
The code below represents a rule whose body consists of three Java method calls: the first to construct a String object, the second to append something to the string, and the third to print the string to the screen.

Listing 3. Prova example

```
hello(Name):-
  S = java.lang.String("Hello "),
  S.append(Name),
  java.lang.System.out.println(S).
```

4.3. Xcerpt

Xcerpt[7] is a Web query language for the "standard Web" (e.g., XML and HTML data) and the Semantic Web (e.g., RDF, Topic Maps, etc.). Xcerpt is "data versatile", i.e. a same Xcerpt query can access and generate, as answers, data in different Web formats. Xcerpt is "strongly answer-closed", i.e. it not only gives rise to construct answers

[3] www.sics.se
[4] www.swi-prolog.org/
[5] www.ifcomputer.com
[6] www.mandarax.org
[7] www.xcerpt.org

in the same data formats as the data queries like, e.g., XQuery [6], but also to further processing in a query program data generated by this same query program. Xcerpt's queries are pattern-based and give rise to incompletely specify the data to retrieve by (1) not explicitly specifying all children of an element, (2) specifying descendant elements at indefinite depths (restrictions in the form of regular path expressions being possible), and (3) specifying optional query parts. Xcerpt's evaluation of incomplete queries is based on a novel form algorithm called "simulation unification". Xcerpt's processing of XML documents is graph-oriented, i.e., Xcerpt is aware of the reference mechanisms (e.g., ID/IDREF attributes and links) of XML. Xcerpt is rule-based. An Xcerpt rule expresses how data queried can be re-assembled into new data items, i.e., an Xcerpt rule corresponds to an SQL view. Xcerpt allows both traversal of cyclic documents and recursive rules, termination being ensured by so-called memo-ing, or tabling, techniques. Xcerpt rules can be chained forward or backward, backward chaining being on the Web the processing of choice. Indeed, if rules can, like Xcerpt's rules, query any Web site, then a forward processing of rule-based programs could require to start a program's evaluation at all Web sites. Xcerpt is inspired from Logic Programming. However, since it does not offer backtracking as programming concept, Xcerpt can also be seen as "set-oriented functional".

Three features of Xcerpt are particularly convenient for querying not only XML but also RDF data. (1) Xcerpt's pattern-based incomplete queries are convenient to collect related resources in the neighborhood of some resources and to express traversals of RDF graphs of indefinite lengths. (2) Xcerpt chaining of (possibly recursive rules) are convenient to express RDFS's semantics, e.g., the transitive closure of the `subClassOf` relation, as well as all kinds of graph traversals. (3) Xcerpt's optional construct is convenient for collecting properties of resources.

5. QUERYING ONTOLOGIES

In this section we describe in principle and examples how to answer the questions from section 3 with Prolog, Prova and Xcerpt.

5.1. Examples solved with Prolog

We assume that relevant parts of the GeneOntology term table and the term2term table are represented in a knowledge base using the following two predicates:

```
1 name2term(N,T). %where N is the name and T the id;
2
3 term2term(R,RT,T1,T2).
4   %where R is the relationship id, RT the ...
        ...relationship type id,
5   %and T1 and T2 are term ids;
6   %and RT=2 for is-a;
```

then the three queries could be written as follows:

Listing 4. Query 1

```
1 % Show the ID of the GO term small GTPase mediated ...
      ...signal transduction?
2 name2term('small GTPase mediated signal ...
      ...transduction', T);
3
4 % Which GTPase  processes are listed in the GO?
5 isDesc('small GTPase mediated signal transduction',...
      ...N);
6
7 isDesc(N1,N2) :-
8     name2term(N1,T1),
9     isa(T2,T1),
10    name2term(N2,T2).
11
12 isa(T,T).
13 isa(T2,T1) :- term2term(_,2,T3,T1), isa(T2,T3).
14
15 isDesc2(N).
16
17 % Which GTPase processes apart from Rho are listed ...
      ...in the GO?
18 isDesc2(N) :-
19     isDesc('small GTPase mediated signal ...
          ...transduction', N),
20     not(isDesc('Rho protein signal transduction',N)...
          ....
```

5.2. Examples solved with Prova

The Prova code very closely resembles the declarative Prolog specification. However, instead of relying on an internal knowledge base, which needs to be loaded entirely into memory, Prova access the GO in a database, which is accessed as needed. In the same way as Prolog, Prova applies backward-chaining to evaluate queries.

The following Prova code first "'imports"' some utility functions, like *dbopen*, which opens a database connection. One database location is provided in line 4. The script evaluates the three statements at once. First, it tries to bind a GO term accession number to a given name, in line 7. The answer is no if the database has no such label for any term. The predicate $name2term$ is defined later in the code (lines 42-45). It opens a database connection to a current GeneOntology MySQL database scheme, constructs the where-clause for the SQL statements in line 44 and issues the SQL query in line 45. The values of the column id in the result set are bound to the variable T.

Second, descendant concepts in the ontology are found for *small GTPase mediated signal transduction* with the predicate $isDesc$. In line 20, GO accession ids are bound to $T1$ if there is a term with the name $N1$ and it has an accession id. Line 21 binds all terms which are sub-concepts of $T1$ to $T2$. For this it uses the recursive definition of isa in line 29 which eventually queries the database using the predicate $isaDB$ defined in line 36. In line 43 the database connection is opened. Line 44 just prepares the where-clause for the SQL statement used in sql_select, line 45.

The third query is similar to the second one but excluding subclasses of *Rho protein signal transduction* in the result.

```
1 :-eval(consult("utils.prova")).
2
3 % Define database location
4 location(database,"GO","jdbc:mysql://server","guest...
      ...","guest").
5
6 % Show the ID of the GO term small GTPase mediated ...
      ...signal transduction?
```

```
:-solve(name2term("small GTPase mediated signal ...
     ...transduction",T)).

% Which small GTPase mediated signal transduction ...
     ...processes are listed in the GO?
:-solve(isDesc("small GTPase mediated signal ...
     ...transduction",N)).

% Which small GTPase mediated signal transduction ...
     ...processes apart from Rho are listed in the GO...
     ...?
:-solve(isDesc2(N)).
isDesc2(N):-
  isDesc("small GTPase mediated signal transduction...
       ...",N),
  not(isDesc("Rho protein signal transduction",N)).

% Defining a descendent
isDesc(N1,N2) :-
  name2term(N1,T1), % N1 has the term id T1
  isa(T2,T1),       % T2 is a T1
  term2name(T2,N2). % T2 has the term name N2

% A term T is a T
isa(T,T).

% Recursive definition of is-a:
% A term T2 is a T1 if T3 is a T1 and T2 is a T3
isa(T2,T1) :-
  isaDB(T3,T1),
  isa(T2,T3).

% This predicate is limited by the number of open ...
     ...connections
% allowed T2 is a T1 if there is a corresponding ...
     ...entry in the
% term2term table of the database
isaDB(T2,T1) :-
  dbopen("GO",DB),
  concat(["term1_id=",T1," and relationship_type_id...
       ...=2"],WhereClause),
  sql_select(DB,term2term,[term2_id,T2],[where, ...
       ...WhereClause]).

% Given the name N, get the term id T
name2term(N,T) :-
  dbopen("GO",DB),
  concat(["name like ",N],WhereClause),
  sql_select(DB,term,[id,T],[where, WhereClause]).

% Given the term id T, get the term name N
term2name(T,N) :-
  dbopen("GO",DB),
  concat(["id=",T],WhereClause),
  sql_select(DB,term,[name,N],[where, WhereClause])...
       ....
}
```

The unmodified output of this is shown below. Note there are three paragraphs. One paragraph per $solve$ statement.

Prova output

```
GO_ID=4280

N="small GTPase mediated signal transduction"
N="Ras protein signal transduction"
N="Rho protein signal transduction"
N="Rac protein signal transduction"
N="regulation of small GTPase mediated signal ...
     ...transduction"
N="regulation of Rac protein signal transduction"
N="negative regulation of Rac protein signal ...
     ...transduction"
N="positive regulation of Rac protein signal ...
     ...transduction"
N="regulation of Rho protein signal transduction"
N="negative regulation of Rho protein signal ...
     ...transduction"
N="positive regulation of Rho protein signal ...
     ...transduction"
N="regulation of Ras protein signal transduction"
N="positive regulation of Ras protein signal ...
     ...transduction"
N="negative regulation of Ras protein signal ...
     ...transduction"
N="positive regulation of small GTPase mediated ...
     ...signal transduction"
N="positive regulation of Rac protein signal ...
     ...transduction"
N="positive regulation of Rho protein signal ...
     ...transduction"
N="positive regulation of Ras protein signal ...
     ...transduction"
N="negative regulation of small GTPase mediated ...
     ...signal transduction"
N="negative regulation of Rac protein signal ...
     ...transduction"
N="negative regulation of Rho protein signal ...
     ...transduction"
N="negative regulation of Ras protein signal ...
     ...transduction"

N="small GTPase mediated signal transduction"
N="Ras protein signal transduction"
N="Rac protein signal transduction"
N="regulation of small GTPase mediated signal ...
     ...transduction"
N="regulation of Rac protein signal transduction"
N="negative regulation of Rac protein signal ...
     ...transduction"
N="positive regulation of Rac protein signal ...
     ...transduction"
N="regulation of Rho protein signal transduction"
N="negative regulation of Rho protein signal ...
     ...transduction"
N="positive regulation of Rho protein signal ...
     ...transduction"
```

Observant readers might notice that the answer to the question "'Which small GTPase mediated signal transduction processes apart from Rho are listed in the GO?"' contains *(negative/positive) regulation of Rho protein signal transduction*. In this query we want the list of processes which are a subclass of *small GTPase mediated signal transduction* but excluding subclasses of *Rho protein signal transduction*. GO currently lists *regulation of Rho protein signal transduction* as *part_of* but not as *is_a Rho protein signal transduction*, see figure 3. However *(negative/positive) regulation of Rho protein signal transduction* are subclasses of *regulation of Rho protein signal transduction* but not of *Rho protein signal transduction*. Therefore the query is in fact answered correctly, but the GO contains an inconsistency in its use of part-of and is-a: *regulation of small GTPase mediated signal transduction* is-a *small GTPase mediated signal transduction* but not *part_of*. We summarize that GO sometimes lists *regulations* of a process as part of the general process and sometimes as being a subclass of it.

5.3. Examples solved with Xcerpt

GeneOntology in XML and RDF. For the GeneOntology two different XML serialization formats and one (inofficial) RDF version are available. The more widespread XML format (referred to in the rest of this paper as GO/XML) actually uses RDF identifiers (URI's and attributes from the RDF namespace) for identifying and referring to terms as the ID/IDREF link mechanism provided in basic XML has been deemed insufficient. This XML format is essentially compatible with RDF/XML [1], the standard serialization of RDF in XML, but extends this format slightly. The second XML format is based upon the OBO syntax for flat files. It differs from GO/XML by using different (non-RDF) identifiers, no use of namespaces, and a slightly simpler structure as it is not based on RDF/XML.

A non-official format of the GeneOntology in standard (non extended) RDF is also available. The main difference to GO/XML is the use of `rdfs:subClassOf` instead of `go:is_a` to represent the subsumption hierarchy among terms and a proper RDF representation of complex information such as database cross-references. This makes processing of this information using standard RDF tools easier.

GeneOntology as Graph and as Triples. Instead of implementing the sample queries from Section 3 on each of these different serialization formats, we propose in the following to define two more abstract views over these concrete serializations using Xcerpt rules:

The first view allows to see the terms and their relations in the GeneOntology as (flat) RDF *triples*. This is similar to the view most RDF query languages such as the W3C's SPARQL [11] provide on the RDF version of the GeneOntology.

Listing 5. RDF Triples for an Xcerpt of the GeneOntology (using N3 [3] notation for RDF triples)

```
1 :GO0007264 rdf:type go:term.
2 :GO0007264 go:name "small GTPase mediated signal ...
      ...transduction".
3 :GO0007264 rdfs:subClassOf :GO0007242.
4 :GO0016601 rdf:type go:term.
5 :GO0016601 go:name "RAC protein signal transduction"...
      ....
6 :GO0016601 rdfs:subClassOf :GO0007264.
7 :GO0007265 rdf:type go:term.
8 :GO0007265 go:name "RAS protein signal transduction"...
      ....
9 :GO0007265 rdfs:subClassOf :GO0007264.
```

The second view allows to view the ontology directly as a *graph* of terms and relations among the terms. This graph view of the GeneOntology is close to the graph view of RDF in Xcerpt as described in [4]: XML and therefore Xcerpt are limited to node-labeled graphs only. The GeneOntology, on the other hand, (just like RDF and other ontology languages) uses a graph model where both nodes (i.e., terms in the ontology) and edges (i.e., relations among the terms) are labeled. To represent such a graph in Xcerpt, labeled edges are represented by labeled nodes with (unlabeled) edges to the source and sink of the original edge. E.g., to express that X stands in part-of relation to Y, there is a part-of subelement for X that contains Y as subelement. This leads to a graph where the children of nodes for terms are nodes for relations and vice versa. Hence, such a representation is often referred to as *striping*.

In the following queries, the Xcerpt compact syntax is used.

View definitions in Xcerpt. The following rule shows how such a graph view can be generated from the GO/XML representation of the GeneOntology:

Listing 6. Graph View on GO/XML

```
1  ns-prefix go="http://www.geneontology.org/dtds/go....
       ...dtd#"
2  ns-prefix rdf="http://www.w3.org/1999/02/22-...
       ...rdf-syntax-ns#"
3
4  CONSTRUCT
5    terms {
6      all var TermAID@term {
7        id { var TermAID },
8        all var Property,
9        optional all var Relation {
10         ^var TermBID
11       }
12     } }
13 FROM
14   go:go {{
15     var TermA → desc go:term {{
16       attributes {{ rdf:about { var TermAID } }},
17       var Property → var Label {{
18         without attributes {{ rdf:resource {{ }} }}
19       }},
20       optional var Relation {
21         attributes {{ rdf:resource { var TermBID } ...
                ...}}
22       }
23     }} }}
24 END
```

An excerpt rule is used, where in the query term (between the **FROM** and **END** keywords) `go:term` elements are matched and bound to the variable `TermA`. The **desc** keyword specifies that these elements may occur at any depth under the root of the XML document with label `go:go`. The query also collects the value of their `rdf:about` attribute (i.e., the ID of the term), all their properties (i.e., sub-elements without `rdf:resource` attribute), and their relations to other terms. Such relations are expressed in GO/XML using sub-elements (labeled, e.g., `go:is_a` or `go:part_of`) with a `rdf:resource` attribute pointing to the related term. The double curly brackets in the query indicate (1) that we do not care about the order among the specified elements and (2) that the query specification is incomplete, e.g., there might be additional sub-elements of the `go:go` document element. The **optional** keyword in line 20 indicates that this part of the query is optional, i.e., a term is also matched, if it has no relations.

In the construct term (between **CONSTRUCT** and **FROM**) the shape of the data constructed by the rule is specified: under the root `terms` for each binding of `TermA` (i.e., for each term in the GeneOntology) a `term` element with the proper ID is created and all its properties are copied from the input data. The crucial part of the construct term are lines 9–11: here for each relation a subelement labeled as in the input is created. This element in turn has as sub-element, the related term. Instead of copying that term, a reference to the term is used indicated by the ^ operator. Such references are defined in line 5 using the @ operator.

A triple view of the GeneOntology can be obtained from the RDF serialization format by using the library for accessing RDF data proposed in [4].

The following view shows how to define the above graph view on such an RDF triple view[8].

Listing 7. Graph View on GO/RDF (same namespace declarations as above)

```
1 CONSTRUCT
2   terms {
3     all var TermID@term {
```

[8] For simplicity, dbxref's that are represented as blank nodes in RDF are not handled.

```
4       id { var TermID },
5       all var Property { var Value },
6       optional all go:part_of {
7         ^var PartOfTermID
8       }
9       optional all go:is_a {
10        ^var IsATermID
11      }
12    }
13  }
14 FROM
15   and {
16     RDF-TRIPLE [
17       var TermID:uri{}, "rdf:type":uri{}, "go:term"...
              ...:uri{}
18     ],
19     RDF-TRIPLE [
20       var TermID:uri{}, var Property:uri{}, literal...
              ... { var Value }
21     ],
22     optional RDF-TRIPLE [
23       var TermID:uri{}, "go:part-of":uri{}, var ...
              ...PartOfTermID:uri{}
24     ],
25     optional RDF-TRIPLE [
26       var TermID:uri{}, "rdfs:subClassOf":uri{}, ...
              ...var IsATermID:uri{}
27     ]
28   }
29 END
```

In essence, the query collects the URIs of all terms in the variable `TermAID` (see lines 2–4 in the **FROM** clause). More precisely, URIs for all instances of `go:term` are collected where instances are expressed using the standard RDF instance relation `rdf:type`. For each such term, all properties and the URIs of all terms it is part or subclass of are collected. Again, there might be no part or subclass relations, therefore the **optional** keyword is used for these conjuncts. The construction is analogous to the case above.

Sample queries in Xcerpt. Query 1 from Section 3 can be expressed in Xcerpt as follows:

```
1  GOAL
2    result {
3      all term-id{
4        var TermID
5      }
6    }
7  FROM
8    terms {{
9      term {{
10       id { var TermID },
11       go:name { "small GTPase mediated signal ...
              ...transduction" }
12     }}
13   }}
14 END
```

In this query we operate on the graph view defined above and query the IDs of all terms with the requested `go:name` property. In the **FROM** clause such IDs are bound to the variable `TermID` using incomplete matching in breadth for both the `terms` and `term` element as both may (and do) have additional sub-elements not specified here.

In the **GOAL** clause a construct term specifying the shape of the final result of the program is given: Using the keyword grouping **all** the IDs of all matched terms are collected, each nested inside its own `term-id` element.

This query can be as easily expressed on the triple view:

```
1  GOAL
2    result {
3      all term-id{
4        var TermID
5      }
6    }
7  FROM
8    and{
9      RDF-TRIPLE [
10       var TermID:uri{}, "rdf:type":uri{}, "go:term"...
              ...:uri{}
11     ],
12     RDF-TRIPLE [
13       var TermID:uri{}, "go:name":uri{}, literal{ "...
              ...small GTPase mediated signal ...
              ...transduction" }
14     ]
15   }
16 END
```

A conjunctions of triples is used to find the IDs of terms that fulfill all properties asked for. Such conjunctions of triples are used often in RDF query languages, e.g., in RDQL [14], SeRQL [5], RQL [8, 9] or the W3C's SPARQL [11]. In SPARQL this query can be expressed as follows:

```
1 SELECT ?TermId
2 WHERE  (?TermID, <rdf:type>, <go:term>),
3        (?TermID, <go:name>, "small GTPase mediated ...
              ...signal transduction")
4 USING  go FOR http://139.91.183.30:9090/RDF/VRP/...
       ...Examples/schema_go.rdf#,
5        rdf FOR http://www.w3.org/1999/02/22-rdf-...
              ...syntax-ns#
```

Query 2, however, is not as easily expressed on such a triple view as it requires recursive traversal of the `rdfs:subClassOf`/ `go:is_a` relation. In fact, none of the above mentioned RDF query languages supports recursive traversal of arbitrary relations and only RQL has specific language constructs for recursive traversal of `rdfs:subClassOf` (as that relation is part of the RDFS standard). In Xcerpt, this query can be expressed very handily on the graph view as follows:

```
1  GOAL
2    result {
3      all term-id{
4        var TermID
5      }
6    }
7  FROM
8    terms {
9      term {
10       id { var TermID },
11       desc(go:is_a|term)* term {
12         go:name { "small GTPase mediated signal ...
              ...transduction" }
13       }
14     }
15   }
16 END
```

Here the Xcerpt's "qualified descendant" construct is used in line 8: all terms are selected that have a term with the requested name as descendant. However, between the descendant and the selected term only `go:is_a` and `term` elements may occur. This ensures that the term is not actually related by the `go:part_of` relation.

On the triple view the query can be expressed as well but requires recursive rules (as in the Prolog and Prova case). For `rdfs:subClassOf` the required rules are contained in the RDFS entailment library developed in [4] (slightly simplified here):

```
CONSTRUCT
 RDFS-TRIPLE[
   var CLASS, "http://www.w3.org/2000/01/rdf-schema#...
        ...subClassOf":uri{}, var SUPERCLASS
 ]
FROM
  and[
    RDF-TRIPLE[
      var CLASS, "http://www.w3.org/2000/01/...
           ...rdf-schema#subClassOf":uri{}, var X
    ],
    RDFS-TRIPLE[
      var X, "http://www.w3.org/2000/01/rdf-schema#...
           ...subClassOf":uri{}, var SUPERCLASS
    ]
  ]
END
```

On this RDFS "view" query 2 can than be easily expressed as follows:

```
GOAL
  result {
    all term-id{
      var TermID
    }
  }
FROM
  and{
    RDF-TRIPLE [
      var TermID:uri{}, "rdf:type":uri{}, "go:term"...
           ...:uri{}
    ],
    RDFS-TRIPLE [
      var TermID:uri{}, "rdfs:subClassOf":uri{}, ...
           ...var X:uri{}
    ],
    RDF-TRIPLE [
      var X:uri{}, "go:name":uri{}, literal{ "small ...
           ...GTPase mediated signal transduction" }
    ]
  }
END
```

Query 3 can be expressed on the graph view as straightforward extension of the previous query:

```
GOAL
  result {
    all term-id{
      var TermID
    }
  }
FROM
  terms {
    term {
      id { var TermID },
      and {
        desc(go:is_a|term)* term {
          go:name { "small GTPase mediated signal ...
               ...transduction" }
        },
        not {
          desc(go:is_a|term)* term {
            go:name { "Rho protein signal ...
                 ...transduction" }
          }
        }
      }
    }
  }
END
```

Notice the use of the **and** keyword to express a conjunction inside of a term. This illustrates another important property of Xcerpt: in contrast to traditional logic programming languages, formulae and terms are not separated, but rather formulae are expressed as terms. In particular, Xcerpt does not distinguish between predicates and terms.

On the triple view it can be expressed as

```
GOAL
  result {
    all term-id{
      var TermID
    }
  }
FROM
  and{
    RDF-TRIPLE [
      var TermID:uri{}, "rdf:type":uri{}, "go:term"...
           ...:uri{}
    ],
    RDFS-TRIPLE [
      var TermID:uri{}, "rdfs:subClassOf":uri{}, ...
           ...var X:uri{}
    ],
    RDF-TRIPLE [
      var X:uri{}, "go:name":uri{}, literal{ "small ...
           ...GTPase mediated signal transduction" }
    ],
    not {
      RDFS-TRIPLE [
        var TermID:uri{}, "rdfs:subClassOf":uri{}, ...
             ...var Y:uri{}
      ],
      RDF-TRIPLE [
        var Y:uri{}, "go:name":uri{}, literal{ "Rho ...
             ...protein signal transduction" }
      ]
    }
  }
END
```

6. COMPARISON OF LANGUAGES AND IMPLEMENTATIONS

Languages. We compare in this article three reasoning-aware query languages and their ability to query biological ontologies. Prolog is the most mature language available in many implementations with a strong support and extensive documentation available. This stands in contrast to the two other relatively new languages, Prova and Xcerpt, currently used by small communities. The two languages are in a developmental state but with finalized specifications for most of the language features and syntax elements. Implementations exist for both of them and are ready to use. The developers give support to interested users. The documentation on both is sufficient to start using the languages.

Access to data sources. Ontologies are available in various formats such as XML, RDF, and databases. The three languages have no principle problem in accessing any of the offered data formats. Xcerpt naturally prefers XML and RDF formats and comes with very versatile querying and construction features for such data making it the best choice for querying ontologies in these formats. The currently available prototype concentrates rather on the language features but on scalability for large ontologies, there Xceprt has difficulties dealing with large ontologies specified in XML. Xcerpt is consciously limited to a single complex data type, viz. XML. Hence, it is only possible to access relational databases by means of XML interfaces to these databases. Direct access to relational databases is not considered for Xcerpt.

Basic Prolog does not offer direct access to XML, RDF or relational databases. But there are implementations such as Quintus Prolog with huge libraries to call C functions and standard UNIX routines.

Prova follows another approach. A programmer familiar with Java might use Prova for parts of the application best expressed with rules, e.g. business rules, workflows, etc. Other components preferably modelled in an object-oriented language remain written in Java. Therefore Prova fully supports database access and can access all XML relevant features supported by Java.

7. CONCLUSIONS

Prolog. Developers of production stage services or applications should use the well-established logic programming language Prolog. Serveral extensions are available which make it possible to externalize task possibly difficult to implement in a pure logic programming language, e.g. database access, GUI programming, graphical visualisation. Prolog clearly has the advantage over Prova and Xcerpt in that it has a large supporting community.
Prova. Prova is the choice of a Java programmer with Prolog experience who aims to develop a system which needs a possibly thin layer of rules to do some reasoning or defining business rules, workflow or agent communication. Prova is available at www.semanticwebrules.org.
Xcerpt. Xcerpt is a general purpose XML and Web query language. It is well-suited for ontology queries over XML and RDF, as well as (X)HTML pages. Due to its relative youth and the focus on language and evaluation theory, a production use of Xcerpt is currently only advisable in small-scale projects. Xcerpt is online at www.xcerpt.org.

Acknowledgement: We would like to thank Alex Kozlenkov, who maintains Prova, and Andreas Henschel for his comments.

8. REFERENCES

[1] Dave Backett. *RDF/XML Syntax Specification (Revised)*. W3C, February 2004.

[2] Sacha Berger, Franois Bry, Oliver Bolzer, Tim Furche, Sebastian Schaffert, and Christoph Wieser. Xcerpt and visXcerpt: Twin Query Languages for the Semantic Web. In *Proc. Int. Semantic Web Conf.*, 11 2004. I4 I3.

[3] Tim Berners-Lee. Notation 3, an RDF language for the Semantic Web. Online only, 2004.

[4] Oliver Bolzer. Towards Data-Integration on the Semantic Web: Querying RDF with Xcerpt. Diplomarbeit/Master thesis, University of Munich, 2 2005.

[5] Jeen Broekstra and Arjohn Kampman. SeRQL: A Second Generation RDF Query Language. In *Proc. SWAD-Europe Workshop on Semantic Web Storage and Retrieval*, 2003.

[6] Don Chamberlin, Peter Fankhauser, Massimo Marchiori, and Jonathan Robie. *XML Query (XQuery) Requirements*. W3C, 2003.

[7] GeneOntologyConsortium. The Gene Ontology (GO) database and informatics resource. *Nucleic Acids Res.*, 1(32):D258–61, 2004.

[8] Gregory Karvounarakis, Sophia Alexaki, Vassilis Christophides, Dimitris Plexousakis, and Michel Scholl. RQL: A Declarative Query Language for RDF. In *Proc. International World Wide Web Conference*, May 2002.

[9] Gregory Karvounarakis, Aimilia Magkanaraki, Sophia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Karsten Tolle. RQL: A Functional Query Language for RDF. In Peter Gray, Peter King, and Alexandra Poulovassilis, editors, *The Functional Approach to Data Management*, chapter 18, pages 435–465. Springer-Verlag, 2004.

[10] Alexander Kozlenkov and Michael Schroeder. PROVA: Rule-based Java-scripting for a bioinformatics semantic web. In E. Rahm, editor, *International Workshop on Data Integration in the Life Sciences DILS*, Leipzig, Germany, 2004. Springer.

[11] Eric Prud'hommeaux and Andy Seaborne. SPARQL Query Language for RDF. Working draft, W3C, 2 2005.

[12] K.D. Pruitt, K.S. Katz, H. Sicotte, and D.R. Maglott. Introducing refseq and locuslink: curated human genome resources at the ncbi. *National Center for Biotechnology Information*, 16(1):44–7, 2000.

[13] Michael Rebhan, Vered Chalifa-Caspi, Jaime Prilusky, and Doron Lancet. Genecards: a novel functional genomics compendium with automated data mining and query reformulation support. *Bioinformatics*, 14(8):656–664, 1998.

[14] Andy Seaborne. RDQL – A Query Language for RDF. www.w3c.org, January 2004.