1 Querying Bio-Ontologies using Xcerpt

In this section, we show a number of ideas for using the Web query language Xcerpt for querying terms and their relations defined in the GeneOntology. Xcerpt is a novel versatile Web query language that provides ample support for querying Web data heterogeneous in structuring and even with regard to representation format used. Xcerpt supports querying of both XML and RDF data, as well as intertwined access to data in these formats.

Xcerpt, a versatile Web query language. Xcerpt [9, 8, 15, 2], cf. `http://xcerpt.org`, is a query language designed after principles given in [7] for querying both data on the "standard Web" (e.g., XML and HTML data) and data on the Semantic Web (e.g., RDF, Topic Maps, etc. data).

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 in the same data formats as the data queries like, e.g., XQuery [10], 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" [6]. 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 "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.

GeneOntology in XML and RDF. For the GeneOntology (at least) two different XML serializations formats and one (unofficial) 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.

Two views on the GeneOntology: Graphs and Triples. Instead of implementing the sample queries from Section [WHERE THE QUERIES ARE DISCUSSED] 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 [13] provide on the RDF version of the GeneOntology.

[SAMPLE DATA IN SOME RDF TRIPLE SYNTAX, E.G. N3]

Listing 1: RDF Triples for an Excerpt of the GeneOntology (using N3 [3] notation for RDF triples)

```
:GO0007264 rdf:type go:term.
:GO0007264 go:name "small GTPase mediated signal transduction".
:GO0007264 rdfs:subClassOf :GO0007242.
:GO0016601 rdf:type go:term.
:GO0016601 Go:name "RAC protein signal transduction".
:GO0016601 rdfs:subClassOf :GO0007264.
:GO0007265 rdf:type go:term.
:GO0007265 go:name "RAS protein signal transduction".
: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*.

[IF YOU HAVE A GRAPH OF SOME SAMPLE DATA OF THE GO, I THINK A PICTURE SHOWING THE STRIPING WOULD BE GREAT HERE]

In the following queries, the Xcerpt compact syntax is used. For a comprehensive description of Xcerpt's compact and XML syntax see [16, 14].

[IF WE HAVE THE SPACE I WOULD LIKE TO SHOW THE SAMPLE DATA IN XML AND AS AN XCERPT DATA TERMS]

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 2: Graph View on GO/XML

```
1  ns-prefix go="http://www.geneontology.org/dtds/go.dtd#"
   ns-prefix rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
3
   CONSTRUCT
5    terms {
       all var TermAID@term {
7        id { var TermAID },
         all var Property,
9        optional all var Relation {
           ^var TermBID
11       }
       } }
13 FROM
     go:go {{
15     var TermA → desc go:term {{
         attributes {{ rdf:about { var TermAID } }},
17       var Property → var Label {{
           without attributes {{ rdf:resource {{ }} }}
19       }},
         optional var Relation {
21         attributes {{ rdf:resource { var TermBID } }}
         }
23     }} }}
   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 an 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 sub-element labeled as in the input is created. This element in turn has as sub-element the related term. Instead of coping 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.[1].

Listing 3: Graph View on GO/RDF (same namespace declarations as above)

```
1  CONSTRUCT
     terms {
3      all var TermID@term {
         id { var TermID },
5        all var Property { var Value },
         optional all go:part_of {
7          ^var PartOfTermID
         }
9        optional all go:is_a {
           ^var IsATermID
11       }
       }
13   }
   FROM
15   and {
       RDF-TRIPLE [
17       var TermID:uri{}, "rdf:type":uri{}, "go:term":uri{}
       ],
19     RDF-TRIPLE [
         var TermID:uri{}, var Property:uri{}, literal { var Value }
21     ],
       optional RDF-TRIPLE [
23       var TermID:uri{}, "go:part-of":uri{}, var PartOfTermID:uri{}
       ],
25     optional RDF-TRIPLE [
         var TermID:uri{}, "rdfs:subClassOf":uri{}, var IsATermID:uri{}
27     ]
     }
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 [SECTION WITH QUERIES] can be expressed in Xcerpt as follows:

```
   GOAL
2    result {
       all term-id{
4        var TermID
       }
6    }
   FROM
8    terms {{
       term {{
10       id { var TermID },
         go:name { "small GTPase mediated signal transduction" }
```

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

```
    }}
  }}
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 (there should be only one, of course). 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 **all** grouping keyword 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:

```
GOAL
  result {
    all term-id{
      var TermID
    }
  }
FROM
  and{
    RDF-TRIPLE [
      var TermID:uri{}, "rdf:type":uri{}, "go:term":uri{}
    ],
    RDF-TRIPLE [
      var TermID:uri{}, "go:name":uri{}, literal{ "small GTPase mediated signal ...
          ...transduction" }
    ]
  }
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 [17], SeRQL [5], RQL [11, 12] or the W3C's SPARQL [13]. In SPARQL this query can be expressed as follows:

```
SELECT ?TermId
WHERE  (?TermID, <rdf:type>, <go:term>),
       (?TermID, <go:name>, "small GTPase mediated signal transduction")
USING  go FOR http://139.91.183.30:9090/RDF/VRP/Examples/schema_go.rdf#,
       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:

```
GOAL
  result {
    all term-id{
      var TermID
    }
  }
FROM
  terms {
    term {
```

```
10      id { var TermID },
        desc(go:is_a|term)* term {
12        go:name { "small GTPase mediated signal transduction" }
        }
14    }
  }
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):

```
1 CONSTRUCT
  RDFS-TRIPLE[
3   var CLASS, "http://www.w3.org/2000/01/rdf-schema#subClassOf":uri{}, var ...
        ...SUPERCLASS
  ]
5 FROM
    and[
7     RDF-TRIPLE[
        var CLASS, "http://www.w3.org/2000/01/rdf-schema#subClassOf":uri{}, var X
9     ],
      RDFS-TRIPLE[
11      var X, "http://www.w3.org/2000/01/rdf-schema#subClassOf":uri{}, var ...
            ...SUPERCLASS
      ]
13  ]
  END
```

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

```
  GOAL
2   result {
      all term-id{
4       var TermID
      }
6   }
  FROM
8   and{
      RDF-TRIPLE [
10      var TermID:uri{}, "rdf:type":uri{}, "go:term":uri{}
      ],
12    RDFS-TRIPLE [
        var TermID:uri{}, "rdfs:subClassOf":uri{}, var X:uri{}
14    ],
      RDF-TRIPLE [
16      var X:uri{}, "go:name":uri{}, literal{ "small GTPase mediated signal ...
            ...transduction" }
      ]
18  }
  END
```

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

```
  GOAL
2   result {
      all term-id{
4       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
```

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] François Bry, Sebastian Schaffert, and Andreas Schröder. A contribution to the Semantics of Xcerpt, a Web Query and Transformation Language. In *Proc. Workshop Logische Programmierung*, March 2004.

[7] Franois Bry, Tim Furche, Liviu Badea, Christoph Koch, Sebastian Schaffert, and Sacha Berger. Querying the Web Reconsidered: Design Principles for Versatile Web Query Languages. *Journal of Semantic Web and Information Systems*, 1(2), 2005. I4.

[8] Franois Bry and Sebastian Schaffert. A Gentle Introduction into Xcerpt, a Rule-based Query and Transformation Language for XML. In *Proc. Int. Workshop on Rule Markup Languages for Business Rules on the Semantic Web*, 2002.

[9] Franois Bry and Sebastian Schaffert. The XML Query Language Xcerpt: Design Principles, Examples, and Semantics. In *Proc. Int. Workshop on Web and Databases*, volume 2593 of *LNCS*. Springer-Verlag, 2002.

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

[11] 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.

[12] 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.

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

[14] Sebastian Schaffert. *Xcerpt: A Rule-Based Query and Transformation Language for the Web*. Dissertation/Ph.D. thesis, University of Munich, 2004.

[15] Sebastian Schaffert and Francois Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. Extreme Markup Languages*, August 2004.

[16] Sebastian Schaffert and Franois Bry. Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In *Proc. Extreme Markup Languages*, 8 2004. I4.

[17] Andy Seaborne. RDQL – A Query Language for RDF. Online only, January 2004.