Xcerpt^{RDF}: A Pattern-based Answer to the Versatile Web Challenge

François Bry[1], Tim Furche[1], Benedikt Linse[1], and Alexander Pohl[1]

Institute for Informatics, University of Munich,
Oettingenstraße 67, D-80538 München, Germany
http://www.pms.ifi.lmu.de/

Abstract. We propose Xcerpt^{RDF}, an extension of the rule based XML query language Xcerpt with language constructs explicitly geared at comfortable querying RDF data, including convenient access to collections, containers, reified statements, and "concise bounded descriptions" for blank nodes. Simulation unification, the formal basis for evaluating Xcerpt queries, is extended to cover the new language constructs and thus to give a formal semantics for Xcerpt^{RDF} queries. In contrast to previous integration approaches such as XSPARQL or GRDDL that combine an XML query language such as XSLT or XQuery with an RDF query language such as SPARQL, Xcerpt^{RDF} requires the user to learn only a single language: most language constructs are sufficiently generic to be applied to both RDF and XML data. Xcerpt^{RDF} is thus a possible solution to the challenge of versatile data access on the Web which has emerged due to the plethora of data formats already online.

1 Introduction

Previous approaches for integrating XML and RDF access fall roughly into two categories: transformation and multi-language approaches. In the former, a pure XML or a pure RDF query language is used and data in the respectively other format can only be accessed by some encoding in the other one. In the latter, a query language for one of the data formats is combined, most often embedded into the other one (e.g., XSPARQL, GRDDL). The advantage of transformation approaches is that users have to learn only a single language. However, this is offset by the need to understand the encoding of RDF in XML or vice versa and very limited support for specifics of the encoded data format that are not present in the native format.

In this article, we take a novel approach to integrating XML and RDF querying: We propose slight extensions to the pattern- and rule-based XML query language Xcerpt that allow convenient querying of RDF and, in contrast to, e.g., SPARQL, address also the graph nature of RDF (cf. Section 5). A large core of language features is shared by both the XML and the RDF version of Xcerpt.
Contributions. More precisely, the paper is organised along the following contributions: **(1)** We underscore the need for pattern based access to RDF data,

in particular to RDF graphs including RDF containers, collections and reification. **(2)** We propose (Section 3) the RDF query language Xcerpt$^{\mathsf{RDF}}$, which, by making use of Xcerpt's pattern based approach including the directives `optional`, `without`, `descendant` is a straight-forward and intuitive extension of Xcerpt. Since it relies on the same evaluation principles as Xcerpt, namely simulation, simulation unification, substitution sets and rule chaining, it is easy to grasp for Xcerpt programmers, and, to a slightly lesser extent to any person that has been involved in logic programming. **(3)** In Section 4 we extend the notion of *simulation* from general semi-structured data to RDF data with its intrinsic particularities, thereby giving a formal semantics to the evaluation of Xcerpt$^{\mathsf{RDF}}$ query terms. **(4)** In Section 5 we address the necessity of support for graph properties in RDF query languages brought up by [1]. We show that all queries described by [1] are expressible in Xcerpt$^{\mathsf{RDF}}$.

2 Preliminaries

The Resource Description Framework (RDF) [2] is considered the foundation for the Semantic Web by encoding information in simple triples of subject, predicate and objects. RDF is a logic based language with a formal model theory [3] relying on URIs to uniquely refer to concepts and real world objects, called *resources* in RDF terminology.

Definition 1 (Basic RDF concepts). *An RDF* triple *or* statement *consists of a* subject*,* predicate *and* object*. The subject may either be a URI, i.e. a global identifier or a* blank node*, i.e. an identifier whose scope is restricted to the containing document. The predicate is always a URI identifying the relation that holds between the subject and the object. The object may either be a URI, blank node, or literal, i.e. a string that represents an atomic value such as number, date, XML fragment, etc. An* RDF graph *is a set of RDF triples.*

Xcerpt [4,5] is a declarative, rule- and pattern-based query language for semi-structured data in general, and XML in particular. In contrast to XPath, Xcerpt patterns, also called *query terms*, allow variables and n-ary queries over trees and graphs. The semantics of query terms is specified in terms of *simulation*, and Xcerpt terms are matched with data by an algorithm called *simulation unification* [5].

3 Xcerpt$^{\mathsf{RDF}}$

In this section, the syntax of Xcerpt$^{\mathsf{RDF}}$ is introduced. We first give a formal definition of Xcerpt$^{\mathsf{RDF}}$ terms, then we intuitively explain the semantics of Xcerpt$^{\mathsf{RDF}}$ containers and collections, reified terms and variables one after another. For the sake of brevity, we omit the query constructs `optional`, `without` and `descendant`, the grouping construct `all` and square brackets, which have the same semantics as in ordinary Xcerpt query terms. Moreover, we omit syntactic sugar for the

RDF/S predicates `rdf:type`, `rdfs:domain` and `rdfs:range`. The interested reader is referred to the full specification of Xcerpt$^{\mathsf{RDF}}$ [6].

Definition 2 (Xcerpt$^{\mathsf{RDF}}$ basic notions). *Xcerpt$^{\mathsf{RDF}}$ terms, predicate-object pairs, containers and collections are recursively defined as follows:*

- *if l is an identifier, then var l, termvar l, and cbdvar l is an Xcerpt$^{\mathsf{RDF}}$* label variable*,* term variable *and* concise bounded description variable*, respectively. Collectively they are referred to simply as Xcerpt$^{\mathsf{RDF}}$ variables and, for a term q, denoted by $Vars(q)$.*
- *an Xcerpt$^{\mathsf{RDF}}$ variable, an RDF blank node, URI or RDF literal is an Xcerpt$^{\mathsf{RDF}}$* term*.*
- *if $t_1, \ldots t_n$ are Xcerpt$^{\mathsf{RDF}}$ terms, then $bagOf\{\{t_1, \ldots, t_n\}\}$, $bagOf\{t_1, \ldots, t_n\}$, $setOf\{\{t_1, \ldots, t_n\}\}$, $setOf\{t_1, \ldots, t_n\}$, $seqOf\{\{t_1, \ldots, t_n\}\}$ and $seqOf\{t_1, \ldots, t_n\}$ are Xcerpt$^{\mathsf{RDF}}$* containers*.*
- *if $t_1, \ldots t_n$ are Xcerpt$^{\mathsf{RDF}}$ terms, then $listOf\{\{t_1, \ldots, t_n\}\}$ and $listOf\{t_1, \ldots, t_n\}$ are Xcerpt$^{\mathsf{RDF}}$* collections*.*
- *if t is an Xcerpt$^{\mathsf{RDF}}$ term, then $< t >$ is an Xcerpt$^{\mathsf{RDF}}$* reified term*.*
- *if t is an RDF term and u is a URI, then $u \rightarrow t$ is an Xcerpt$^{\mathsf{RDF}}$* predicate object pair*.*
- *if N is an RDF blank node or URI, and each t_i in $t_1, \ldots, t_n$ is either an Xcerpt$^{\mathsf{RDF}}$ predicate object pair, an Xcerpt$^{\mathsf{RDF}}$ container or collection, or a reified term, then $N\{t_1, \ldots, t_n\}$ and $N\{\{t_1, \ldots, t_n\}\}$ are Xcerpt$^{\mathsf{RDF}}$ terms.*

$Vars(t)$ ($Child(t)$) denotes the set of variables (children) of a term t.

An Xcerpt$^{\mathsf{RDF}}$ term containing no variables and only single curly braces is called an Xcerpt$^{\mathsf{RDF}}$ *data term*. Each Xcerpt$^{\mathsf{RDF}}$ data term can be uniquely mapped to an RDF graph. An Xcerpt$^{\mathsf{RDF}}$ term containing only single curly braces is called an Xcerpt$^{\mathsf{RDF}}$ *construct term*. Hence, every Xcerpt$^{\mathsf{RDF}}$ data term is also an Xcerpt$^{\mathsf{RDF}}$ construct term. The notion of an Xcerpt$^{\mathsf{RDF}}$ *query term* is the most general one and coincides with the one of Xcerpt$^{\mathsf{RDF}}$ terms. Query terms appear in the body of Xcerpt$^{\mathsf{RDF}}$ rules, while construct terms appear within their heads.

An RDF triple `_:A eg:b "c"`, where `_:A` is a blank node, `eg:b` is a URI, and `"c"` is an RDF literal, is represented in Xcerpt$^{\mathsf{RDF}}$ as the term `_:A{ eg:b→ "c" }`. The arrow-syntax is chosen to convey the graph nature of RDF data. Literals, blank nodes, and URIs are represented the same way as in other popular RDF query languages such as SPARQL and RQL.[1] Terms may be nested (e.g. `eg:a{ eg:b→ eg:c{ eg:d→ eg:e } }` or grouped by subject (e.g. `eg:a{ eg:b→ eg:c, eg:d→ eg:e }`. [2].

[1] URIs are commonly abbreviated using a namespace prefix. The prefix `eg` is assumed to be bound to the namespace `http://example.org` in the following.

[2] To further reduce the verbosity of the representation of RDF graphs as Xcerpt$^{\mathsf{RDF}}$ terms, also grouping by predicate, object, by subject and predicate, predicate and object, and by subject and object is supported. For the sake of brevity we refer the interested reader to [6]

Since RDF graphs may be unconnected or have multiple root nodes, they generally cannot be represented by a single Xcerpt$^{\mathsf{RDF}}$ term, but only by a conjunction of Xcerpt$^{\mathsf{RDF}}$ terms. We therefore informally introduce the notion of an Xcerpt$^{\mathsf{RDF}}$ graph, that covers all RDF graphs. The statements "anna knows bob" and "chuck knows bob" are written in Xcerpt$^{\mathsf{RDF}}$ using the FOAF[3] vocabulary and appropriate URIs as the following Xcerpt$^{\mathsf{RDF}}$ graph:[4]

```
RDFGRAPH{ anna{ foaf:knows→ bob}, chuck{ foaf:knows→ bob} }
```

3.1 Representation and Querying of RDF Containers and Collections

As in any other data description formalism, also in RDF there is a need for describing not only single valued, atomic resources, but groups of resources or complex, structured objects. This need is in conflict with one of the most basic design principles of RDF, namely the one of encoding any information within simple statements made up of atomic subjects, predicates and objects. RDF solves this conflict by a canonical encoding of complex structures (such as sets or lists) in RDF triples, at the cost of sacrificing their "intuitive" semantics:

Containers and Collections in RDF and Xcerpt$^{\mathsf{RDF}}$. RDF supplies a predefined vocabulary for RDF Containers and Collections. An RDF container is either a bag, sequence or a set, and is represented by a set of triples involving the predicates `rdf:_1`, `rdf:_2`, `rdf:_3`, ..., and the classes `rdf:Bag`, `rdf:Seq` or `rdf:Set`. RDF sequences have the intuitive semantics of being ordered, RDF bags are unordered and may contain the same element more than once, whereas RDF sets are also unordered, but should not have duplicate elements. The RDF model theory [3], however, does not enforce this semantics. Therefore, an RDF resource (i) may be typed as an RDF bag and an RDF sequence at the same time, (ii) may contain multiple statements involving the predicate `rdf:_i` for some integer i, or (iii) may be typed as an RDF set, but include the same element more than once, and there would be still valid RDF interpretations for the RDF graph containing this resource.

In Xcerpt RDF we **(1)** provide convenient syntax for containers, collections, etc. and **(2)** enforce the intuitive semantics of theses concepts. However, the user can vote to disregard the specific constructs for these RDF constructs and still represent RDF graphs violating their intuitive semantics in Xcerpt$^{\mathsf{RDF}}$ by the same triple encoding as in RDF.

A shopping cart with the entries `eg:milk` and `eg:coffee` represented by an RDF bag is given by the following triples, and the equivalent short hand notation in Xcerpt$^{\mathsf{RDF}}$:

[3] urlwww.foaf-project.org/

[4] The keyword `RDFGRAPH` serves to group several Xcerpt$^{\mathsf{RDF}}$ terms in a graph. Xcerpt$^{\mathsf{RDF}}$ also supports named graphs; for details see [6].

```
eg:cart123 rdf:type rdf:Bag .
eg:cart123 rdf:_1 eg:milk .
eg:cart123 rdf:_1 eg:milk .
eg:cart123 rdf:_2 eg:coffee .
```

```
eg:cart123{ bagOf{
  eg:milk, eg:milk,
  eg:coffee
} }
```

RDF sets and sequences are written in the same way with the keywords `seqOf` and `setOf`. Xcerpt$^{\mathsf{RDF}}$ not only provides shorthand notations for *representing* RDF graphs including containers, but also for *querying* such graphs.

`eg:cart123{ bagOf{{ var Item }} }` is a query that binds each item in the shopping cart to the variable `Item`. The query matches with the data above and yields the bindings `eg:milk`, and `eg:coffee`. Substituting double curly braces by single ones in the query would result in a query that only matches with RDF bags containing a single item.

RDF Containers may only be used to state that some resource r_m is member of some other resource r_c, but cannot be used to state that there are no *other* members r_o of r_c.

In contrast, RDF collections are used to model data that is completely specified, i.e. closed, and are written as RDF statements involving the vocabulary `rdf:List`, `rdf:first` and `rdf:rest`. The Xcerpt term in Listing 1.1 corresponds to Fig. 16 of [7] and asserts that Amy, Mohamed and Johann are the only students of the course 6.001.

Again, the RDF model theory does not enforce the intuitive semantics of closedness of RDF collections. An RDF graph may contain a node that is subject of multiple statements with the predicate `rdf:first` or `rdf:rest`, and a container may also be unclosed. A model theory that formalizes the intuitive semantic extensions brought to RDF by containers and collections has, to the best of our knowledge, not yet been specified.

```
eg:courses/6.001 {
  eg:students/vocab/#students{
    listOf [
      eg:students/Amy,
      eg:students/Mohamed,
      eg:students/Johann ] } }
```

Listing 1.1. Shorthand notation for RDF lists in Xcerpt numbers

```
desc var List as
  listOf[[
    eg:students/
      Mohamed
  ]]
```

Listing 1.2. Finding all Lists with eg:Mohamed

In accordance with the intuitive semantics of RDF containers and collections, Xcerpt$^{\mathsf{RDF}}$ only allows one of the keywords `bagOf`, `setOf`, `seqOf` and `listOf` as a child of a term. Moreover, if one of these keywords is used to describe a resource, there must not be any other statements describing the same resource and involving the vocabulary `rdf:Bag`, `rdf:Set`, `rdf:Seq`, `rdf:List`, `rdf:_1`, `rdf:_2`, etc. Hence the term `a{ bagOf{ b }, rdf:type→ c}` is *not* a valid Xcerpt$^{\mathsf{RDF}}$ term. As a result of this convention, any RDF graph that can be represented as an Xcerpt$^{\mathsf{RDF}}$ term making use only of the syntactic sugar key words for representing RDF containers and collections, is guaranteed to respect the intuitive semantics. Nevertheless, there is still the possibility of representing arbitrary RDF graphs by writing the RDF triples that make up a container or a collection directly as

an Xcerpt^{RDF} term, thereby doing without the keywords. Listing 1.2 shows an Xcerpt^{RDF} query that matches with all RDF graphs containing an RDF list with the entry `eg:students/Mohamed`.

3.2 Representation and Querying of Reified Triples

Reification is an RDF mechanism to encode information about statements, in other words to make RDF *meta statements*. For this purpose, RDF provides a reification vocabulary consisting of the RDF predicates `rdf:subject`, `rdf:predicate` and `rdf:object` and the RDF resource `rdf:Statement`. The Xcerpt^{RDF} term in Listing 1.3 states that bob believes that anna likes him. Listing 1.4 gives the corresponding shorthand notation. The query below selects all reified statements with the predicate `eg:likes`.

```
eg:bob{ eg:believes→ _:St{
  rdf:type→ rdf:Statement,
  rdf:subject→ eg:anna,
  rdf:predicate→ eg:likes,
  rdf:object→ eg:bob
}
```

Listing 1.3. RDF reification

```
eg:bob{
  eg:believes→ _:St{
    <eg:anna{
      eg:likes→ eg:bob }>
  }
}
```

Listing 1.4. Reification shorthand

```
desc var Statement {{ < var _ {{ eg:likes→ var _ }} }}
```

Just as with the RDF container and collection vocabulary, the RDF model theory does not associate any formal semantics with the reification vocabulary. The intuitive semantics suggest that there must not be two distinct statements with the predicate `rdf:subject`, `rdf:predicate`, or `rdf:object` originating from the same node of an RDF graph. Secondly, if a node is subject of a statement with one of these three properties, it must also be subject of statements with the other two properties and be typed as an `rdf:Statement`. Third, all resources appearing as the object of a statement with predicate `predicate` must be of type `rdf:Property`. While RDF graphs that do not agree with these conventions are perfectly valid under the RDF model theory, they cannot be represented by Xcerpt^{RDF} terms that use the Xcerpt^{RDF} shorthand syntax for reification. In other words, every Xcerpt^{RDF} term that only uses the shorthand syntax for asserting reified statements agrees with the intuitive semantics of RDF reification. Still, arbitrary RDF graphs can be encoded in Xcerpt^{RDF} making use of the unabbreviated syntax.

3.3 Xcerpt^{RDF} Programs, Rules, Label, Term/Graph and CBD variables

An Xcerpt^{RDF} program is a set of Xcerpt^{RDF} rules. An Xcerpt^{RDF} rule consists of a query part and a construct part. Just as in logic programming, Xcerpt^{RDF} programs may be evaluated in a forward or backward chaining manner. During a forward chaining evaluation of an Xcerpt^{RDF} program the query part of a rule is

evaluated first, resulting in a set of variable assignments (an assignment includes one binding for each variable in the query), and subsequently this set is applied to the construct part of the rule.

During the evaluation of an Xcerpt query, values for variables are determined by an algorithm called *simulation unification*[5]. Variables serve as a means of carrying over nodes or subgraphs from the data being queried to the result to be computed. A set of bindings for a set of distinct variables is called a *substitution* and a set of substitutions for the same set of variables is called a *substitution set*. As mentioned above, Xcerpt$^{\mathsf{RDF}}$ differentiates between three different kinds of variables: label variables, term variables and variables for concise bounded descriptions.

Label variables are bound to single atomic values, i.e. to URIs, blank nodes or RDF literals. Depending on their appearance within an Xcerpt$^{\mathsf{RDF}}$ query term, they may either be bound to subjects, predicates or objects of RDF statements. When authoring Xcerpt$^{\mathsf{RDF}}$ programs, care must be taken that label variables appearing in object position in the query also appear in object position in the construct term of a rule. Otherwise, the application of the substitution set computed by evaluating the query term to the construct term of the rule may result in RDF graphs that have literals in subject position, which are invalid according to the RDF data model. Similar constraints hold for blank nodes. It can be easily determined at compile time whether an Xcerpt$^{\mathsf{RDF}}$ rule is safe in the sense of only generating valid RDF graphs.

In contrast to label variables, the bindings for graph variables are not single atomic values, but entire RDF subgraphs rooted at the node of the RDF graph that is matched by the node in the query that carries the graph variable. Care must be taken when using graph variables, since in the case of densely interconnected RDF graphs, graph variables may be bound to the entire data. In practical applications such as FOAF or DOAP documents, this is, however, rarely the case.

Variables for *concise bounded descriptions* (CBD) are a trade-off between label variables and graph variables. A CBD of a resource is "a general and broadly optimal unit of specific knowledge about that resource to be utilized by, and/or interchanged between, semantic web agents" [8]. For space reasons, we do not reproduce the exact definition of CBDs, but rather refer to [8]. Xcerpt$^{\mathsf{RDF}}$ treats CBD-variables in the same way as graph variables.

4 Term Simulation including Containers, Collections and Reification

Ground query term simulation is a method for formally specifying the semantics of a pattern based query language, and has first been proposed in [5] for giving a semantics to Xcerpt query terms on semi-structured data. Simulation formalizes the intuitive notion of *matching* a pattern with some data. Simulation is designed to be a transitive relation and to coincide with the subsumption relationship –

i.e. a term t_1 simulates with another term t_2 iff t_1 subsumes t_2 under simulation. This property of simulation has recently been shown in [9].

In this section we extend the definition of simulation from Xcerpt terms over XML data to Xcerpt terms over RDF data. For the sake of brevity we only provide the definition for terms excluding the constructs `desc`, `optional` and `without`. The interested reader is referred to [6].

Definition 3 (Ground Xcerpt$^{\mathsf{RDF}}$ term simulation). *Let q_1 and q_2 be two ground Xcerpt$^{\mathsf{RDF}}$ terms. A relation $\mathcal{S}$ is a simulation of q_1 into q_2, if the following conditions hold.*

- *$(q_1, n) \in \mathcal{S}$ for some node n in q_2.*
- *if $(q, q') \in \mathcal{S}$, then there must be some injective mapping π from $Child(q)$ to $Child(q')$ such that $(c, \pi(c)) \in \mathcal{S}$. If q has breadth-complete subterm specification, π must be bijective.*
- *if $(q, q') \in \mathcal{S}$ and q is typed as an RDF bag by a subterm b using the* bagOf *keyword, then also q' must be typed as an RDF bag by a term b' using the* bagOf *keyword such that $(b, b') \in \mathcal{S}$. The same holds for the keywords* setOf, seqOf *and* listOf.
- *if $(q, q') \in \mathcal{S}$ and q is of the form $bagOf\{\{\ldots\}\}$, then also q' must start with the* bagOf *keyword, and there must be some injective mapping π from the elements $e_1, \ldots e_n$ in q to the elements in q' such that $(e_i, \pi(e_i)) \in \mathcal{S}$ for all $1 \leq i \leq n$. If q is breadth-complete, then the mapping π must be bijective. Analogous conditions hold for the cases where q starts with the other keywords for RDF collections and containers.*
- *if $(q, q') \in \mathcal{S}$ and q has some subterm s of the form $< t >$ – i.e. a reified subterm – then q' must have some reified subterm s' of the form $< t' >$ with $(t, t') \in \mathcal{S}$.*

We say that an Xcerpt$^{\mathsf{RDF}}$ q simulates into an Xcerpt$^{\mathsf{RDF}}$ term q' if there is a simulation relation between q and q'. A (not necessarily ground) Xcerpt$^{\mathsf{RDF}}$ term q simulates into an Xcerpt$^{\mathsf{RDF}}$ term q' if there is a substitution $\sigma : vars(q) \to \mathcal{D}$ such that all substitutions $\tau : vars(q') \to \mathcal{D}$ satisfy $\tau \circ \sigma(q)$ simulates into $\tau(q')$.

5 Support for Graph Properties in Xcerpt$^{\mathsf{RDF}}$

This section sheds light on the expressiveness of Xcerpt$^{\mathsf{RDF}}$ query terms and programs. Recently[1], the need for support of graph properties in RDF query languages was highlighted, and a set of queries that should be easily formulated and answered by an expressive RDF query language was outlined. In this section we present solutions to all seven of these problems in Xcerpt$^{\mathsf{RDF}}$, thereby underscoring the ease of reusing parts of an Xcerpt program, which is due to the declarative, rule-based nature of Xcerpt$^{\mathsf{RDF}}$.

- A query which is very frequently asked according to [1] is the one of finding all nodes adjacent to some resource `eg:a`. The following Xcerpt$^{\mathsf{RDF}}$ query, which

is a disjunction of two Xcerpt^{RDF} query terms, is a solution to this problem and returns a unary substitution set, i.e. a substitution set including bindings for only one variable. The variable `var _` is an anonymous label variable, and does not induce any variable bindings, but merely serves as a place-holder.

```
or(var R{{ var _ --> eg:a }}, eg:a{{ var _ --> var R }})
```

Finding all predicates of statements involving the resource `eg:a` is solved in a very similar manner, but is omitted for the sake of brevity.

– The third problem of finding the degree of a given resource `eg:a` – i.e. the number of statements involving the resource – is solved by the following rule which is answer closed in the sense that the answer to this rule is again an RDF graph[5], and which makes use of the aggregate function `&count` for computing the degree. The keywords `CONSTRUCT` and `FROM` delimit the head of the rule; the rule body is found between the keywords `FROM` and `END`.

```
CONSTRUCT eg:a{ eg:degree--> &count(all var P) }
FROM or(var _{{var P-->eg:a}},eg:a{{var P-->var _}}) END
```

– The following multi-rule program finds all directed paths between the resources `eg:a` and `eg:b`. The first rule is the base case for recursively computing the extension of the predicate `from_to_path` that represents all paths in the RDF graph starting at `eg:a`, and which is represented as an ordinary Xcerpt data term[6], and the second rule is the recursive case. Finally, the third rule selects only those paths that end at `eg:b`.

```
CONSTRUCT from_to_path[ eg:a, var To,
  path[ eg:a, path[ var P, path [ var To, nil ] ] ] ]
FROM eg:a{{ var P --> var To }} END
CONSTRUCT from_to_path[ var From, var To,
  path[ var P, path[ var To, var Path ] ] ]
FROM and ( from_to_path [ var From, var To1, var Path ],
  var To1{{ var Pred--> var To }} )                    END
GOAL     paths[ all var Path ]
FROM     from_to_path[ eg:a, eg:b, var Path ]         END
```

– The fifth problem of finding the distance between two given resources within an RDF graph is easily composed by reusing the previous solution and by making use of the aggregate function `&count` and the grouping construct `all`.

```
CONSTRUCT distance[
  var From, var To, var Path, &count(all var Node) ]
FROM
  from_to_path[ var From, var To,
    var Path as path [ desc var Node ]  ]            END
```

– The sixth problem of finding fixed length paths between two given resources `eg:a` and `eg:b` is a simple view on the data computed in the previous query:

[5] Note that all Xcerpt^{RDF} rules, and thus also programs, are answer closed in the sense that they generate either RDF graphs or XML documents

[6] Xcerpt data terms are isomorphic to fragments of XML documents

```
GOAL      short_paths[ eg:a, eg:b, all var Path ]
FROM      distance[ eg:a, eg:b, var Path, 2 ]          END
```

– Also the last problem of finding the diameter, i.e. the maximum length of paths within an RDF graph is formulated as a simple view making use of the aggregate function `max`.

```
GOAL    diameter[ &max(all var Depth) ]
FROM    distance[ var _, var _, var _, var Depth ]    END
```

6 Conclusion

Xcerpt$^{\mathsf{RDF}}$ is a an answer to the challenge which many developers of data intensive applications on the Web are confronted with, and which is brought up by the Semantic Web: the easy querying, transformation and reasoning with both XML and RDF data. Future work includes the following goals:

(1) RDF support must be integrated into the prototype. **(2)** In order to ensure termination of a larger set of programs, tabling of subgoal solutions must be integrated into the prototype. **(3)** Query subsumption is a well known optimization issue in rule based query languages. Recently, subsumption between Xcerpt queries has been shown to be decidable [9]. An algorithm for deciding subsumption must be incorporated into the tabling engine mentioned above. **(4)** Xcerpt's semantics for programs including negation as failure and grouping constructs is currently limited to negation and grouping stratifiable programs. It turns out, however, that many sensible Xcerpt programs do not necessarily fall into this category, and that the notion of *local stratification* can be adapted to logic programs involving a more involved kind of unification (such as simulation unification). We are currently working on extending Xcerpt's formal semantics to include locally negation and grouping stratifiable programs.

References

1. Angles, R., Gutierrez, C.: Querying RDF Data from a Graph Database Perspective. In: Proc. European Semantic Web Conf. (ESWC). Volume 3532 of LNCS. (2005)
2. Klyne, G., Carroll, J.J.: Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation, W3C (February 2004)
3. Hayes, P.: RDF Semantics. W3C Recommendation, W3C (2004)
4. Schaffert, S., Bry, F.: Querying the Web Reconsidered: A Practical Introduction to Xcerpt. In: Proc. Extreme Markup Languages. (2004)
5. Schaffert, S.: Xcerpt: A Rule-Based Query and Transformation Language for the Web. PhD thesis, University of Munich (2004)
6. Pohl, A.: RDF Querying in Xcerpt: Language Constructs and Implementation. Deliverable I4-Dx2, REWERSE (2008)
7. Manola, F., Miller, E.: RDF Primer. W3C Recommendation, W3C (February 2004)
8. Stickler, P.: CBD—Concise Bounded Description. W3C Submission, Nokia (2005)
9. Bry, F., Furche, T., Linse, B.: Simulation Subsumption or Déjà vu on the Web. In: Proc. Int'l. Conf. on Web Reasoning and Rule Systems (RR). (2008)