I4-DX2

RDF Querying in Xcerpt

Project title:	Reasoning on the Web with Rules and Semantics
Project acronym:	REWERSE
Project number:	IST-2004-506779
Project instrument:	EU FP6 Network of Excellence (NoE)
Project thematic priority:	Priority 2: Information Society Technologies (IST)
Document type:	D (deliverable)
Nature of document:	R (report)
Dissemination level:	PU (public)
Document number:	IST506779/Munich/I4-DX2/D/PU/a1
Responsible editors:	Benedikt Linse
Reviewers:	Tim Furche and François Bry
Contributing participants:	Munich
Contributing workpackages:	I4
Contractual date of deliverable:	—
Actual submission date:	10 April 2008

Abstract

A versatile query language provides capabilities of querying ordinary XML based data as well as Semantic Web data and thus builds a bridge between both worlds. Xcerpt, a pattern-based deductive query and transformation language for semi-structured data, shows to be a perfectly appropriate candidate to become such a versatile query language.

This thesis investigates and develops new language constructs for Xcerpt which allows convenient querying of RDF graphs. These language constructs also comprise numerous shorthand notations for specific RDF constructs such as containers, reification and concise bounded descriptions.

Besides the design of the new syntax, this thesis also investigates how RDF queries can be evaluated in Xcerpt. It is thereby shown that Xcerpt's simulation unification is a very well-suited means to cope with RDF graphs: On the one hand, it can be easily modified and adapted for RDF. On the other hand, simulation unification can be used for a variety of tasks: the evaluation of RDF queries, the optimization of query evaluation by determining subsumption and finally the checking of leanness of RDF graphs.

Keyword List

language design, RDF, Xcerpt, syntax, constructs

Project co-funded by the European Commission and the Swiss Federal Office for Education and Science within the Sixth Framework Programme.

RDF Querying in Xcerpt

Alexander Pohl[1]

[1] Institute for Informatics, University of Munich, Germany
http://pms.ifi.lmu.de/

10 April 2008

Abstract
A versatile query language provides capabilities of querying ordinary XML based data as well as Semantic Web data and thus builds a bridge between both worlds. Xcerpt, a pattern-based deductive query and transformation language for semi-structured data, shows to be a perfectly appropriate candidate to become such a versatile query language.

This thesis investigates and develops new language constructs for Xcerpt which allows convenient querying of RDF graphs. These language constructs also comprise numerous shorthand notations for specific RDF constructs such as containers, reification and concise bounded descriptions.

Besides the design of the new syntax, this thesis also investigates how RDF queries can be evaluated in Xcerpt. It is thereby shown that Xcerpt's simulation unification is a very well-suited means to cope with RDF graphs: On the one hand, it can be easily modified and adapted for RDF. On the other hand, simulation unification can be used for a variety of tasks: the evaluation of RDF queries, the optimization of query evaluation by determining subsumption and finally the checking of leanness of RDF graphs.

Keyword List
language design, RDF, Xcerpt, syntax, constructs

Contents

Overview of this Deliverable

Xcerpt is the rule-based query language is at the heart of the REWERSE project. Its pattern-based approach to querying semi-structured and graph-structured data has proven to be intuitive to the user and a declarative way of writing reusable and easy to understand programs. One of the declared goals of developing Xcerpt is its data versatility. Since XML has become the de facto lingua franca for representing data on the web, and for Xcerpt's ability to natively deal with XML Xcerpt already can treat all Web formats that have some kind of XML serialization such as HTML, RDF, Topic Maps, microformats, etc. RDF as one of these formats is prone to become one of the basic building blocks for the semantic web. Besides its possible serialization as XML, RDF also offers an abstract graph data model, that is distinct from its XML serialization and is the recommended way of keeping RDF data in mind. XML query languages such as XQuery, XSLT, and also Xcerpt have been used to syntactically process the XML serializations of RDF data, but this approach has often ignored the underlying semantics of RDF. Xcerpt^{RDF} is an attempt to overcome this limitation, by adhering to the true graph data model of RDF, providing a syntax aimed at RDF specificities such as RDF containers, collections, reifications, blank nodes and concise bounded descriptions. While Xcerpt^{RDF} is a somewhat ambitious extension to Xcerpt, it builds upon Xcerpt's strengths such as simulation unification, chaining, rich query patterns, and extends its possibilities in a straightforward and coherent manner. The outcome is a versatile query language that is especially useful for the plethora of use-cases that depend on querying both ordinary and Semantic Web data.

In the following deliverable we discuss the latest revision of RDF access in Xcerpt. In contrast to previous approaches (as described in deliverable I4-D6 and [7]), this revision

1. introduces language constructs specifically for querying RDF data.
2. shows that simulation unification can be leveraged to check and ensure leanness of RDF graphs,
3. adapts the Xcerpt data model to properly fit RDF graphs.

Chapter 1

Introduction

The Resource Description Framework (cf. [4, 17, 19, 15, 9]), or RDF for short, is one of the cornerstones of Semantic Web technology. It is a formal language that enables the provision of meta-data, i.e. data that describe conventional web data. Additionally, RDF can be used for making statements about real world entities that are not existent or retrievable in the world wide web, for example persons or organisations. In RDF, one can make statements about anything that is identifiable by a URI. Thus, RDF is capable of providing meta-data for the conventional web as well as data that is unrelated to conventional web data, for example social networks as they are established in the FOAF project [10] – one of several RDF applications. Semantic Web and conventional web data are often associated with each other by kind of references (for example, using the link-Element in HTML, cf. [27]) or by mixing both conventional and Semantic Web data (for example, RDF in the head section of an HTML document [27], RDFa [8, 1], Microformats [26]). Apart from these "explicit" associations, both worlds also come together in query scenarios. As an example, consider the following case: The names of all people of a FOAF network whose email addresses are listed in a given HTML page should be extracted and the relationships between the persons to which the names refer to should be rendered as an SVG graphic (cf. [2]). Usually, in order to support both data formats, this task involves a query processor for the email addresses in the HTML page, a query processor for the RDF document representing the FOAF network and a transformation processor for the creation of the SVG graphic out of the query results. Today's standard query languages are not capable of maintaining such data interrelations, since they can only query either conventional web data or Semantic Web data so that several kinds of query languages and transformation languages must be used simultaneously which is inconvenient and tedious. A *versatile* query language, i.e. a query language that is able to process both conventional and Semantic Web data, could remedy this situation. In supporting both data models, a versatile query language provides a high flexibility in processing web data and in offering common language constructs for both worlds to a great extent; it provides a high convenience and ease in tackling sophisticated querying of both the conventional and the Semantic Web.

This thesis describes such a versatile query language designed for both semi-structured data (like XML or HTML) and RDF data. Furthermore, this query language provides complex constructions allowing the structuring of answers as desired. This thesis extends the capabilities of the existing Xcerpt [22] in a way that one can also query and transform RDF data.

In this thesis the versatile query language, which combines the semi-structured query language with its extension to query RDF, is referred to as Xcerpt. The original language as described in [22] only being capable of querying and transforming semi-structured data is referred to as Xcerpt^{XML}. Finally, the component of Xcerpt for querying and transforming RDF data introduced in this thesis is referred to as Xcerpt^{RDF}. Figuratively, the following equation illustrates the relation between the three languages:

$$\boxed{\text{Xcerpt} = \text{Xcerpt}^{XML} + \text{Xcerpt}^{RDF}}$$

1.1 Extending Xcerpt^{XML} for Querying RDF

In [12] it has been argued that a versatile query language shall exhibit the following three features at its core.

Convenient and efficient access to any kind of web data A versatile query language should be able to process both conventional and Semantic Web data disregarding the way they are made available (e.g. separated in different documents or mixed within a document, serialization format).

Referential Transparency and answer-closedness A versatile query language should provide referential transparency (i.e. values only depend on their environments and not on the point of time in which they are evaluated) and be answer-closed (i.e. querying XML yields answers compliant to the data model of XML, querying RDF yields answers compliant to the data model of RDF) which is realized by rules and patterns. Patterns serve as molds for valid data in which variables are replaced by values. Rules provide basic reasoning capabilities.

Admissibility of incomplete queries and answers It should be possible to specify the requested information only partially and to select only important pieces of information from given answers. This should be admissible due to the heterogeneity and incompleteness of web data.

Xcerpt^{XML} as a declarative and pattern-based query and transformation language for semistructured data is referentially transparent, answer-closed and admits incomplete query specifications as well as incomplete answers. The retention of these features for an extension to query RDF makes Xcerpt^{XML} a versatile query language. For the reasons given below, Xcerpt^{XML} is amenable to extensions that are necessary in order to master the challenges that accompany the extension for dealing with RDF. These challenges, among others, comprise the following (for a detailed discussion refer to [13]).

Data Model The data model of XML is basically a node-labelled tree (i.e. a rooted acyclic graph) with additional edges expressed using the attributes ID and IDREF possibly yielding non-tree graphs. The data model of RDF is an unrooted graph with labelled nodes and edges. Since the data model of Xcerpt^{XML} is based on semi-structured graphs Xcerpt^{XML} can be adapted to RDF more easily than other query languages. Besides the graph nature

of Xcerpt^{XML}'s data model, labelled edges as in the RDF data model can also be transformed to unlabelled edges in introducing nodes representing these labels, thus bringing the data models of XML and RDF closer to each other and paving the way to a common query language.

Incomplete and Unbounded Data XML data is complete whereas RDF data is incomplete, so to speak. In contrast to the XML data model RDF does not distinguish between occurrences of resources. That is, a resource can be addressed globally, hence allowing anyone to make statements about a resource. As a consequence, it cannot be assumed that the knowledge about a given resource is completely given. However, several scenarios are conceivable in which completeness of data and restricted access to RDF data is required. As a means to restrain incompleteness of data, named graphs [14] have shown to be useful in practice.

Many features which are necessary to cope with RDF and other Semantic Web data are already available in Xcerpt^{XML}, thus, it is worthwile to undertake the expenditure of extending Xcerpt^{XML} for querying RDF:

- Xcerpt^{XML} has the capability of to cope with graph based data.
- Subterms can be specified as unordered.
- Xcerpt^{XML} rules provide means for RDF entailment.
- The semantics of Xcerpt^{XML} is intuitive and can be easily extended for RDF querying.

1.2 Contributions and Outline of this Thesis

This thesis discusses and suggests syntactic and semantic extensions to Xcerpt^{XML}. Xcerpt^{XML} extended with RDF querying capabilities will be denoted as Xcerpt^{RDF} in this thesis. An overview of existent RDF query languages is given in order to recognize language constructs being useful and convenient. These query languages do not only serve as positive examples. In cases of insufficient constructs, alternative solutions are suggested and discussed. Xcerpt^{RDF} is provided with a declarative semantics that is tailored for querying RDF. Again, other query languages serve as examples. Furthermore, this thesis aims at finding and discussing solutions for problems regarding querying RDF that are currently not or only marginally solved by existent query languages:

- An intuitive syntax that integrates well to the existing Xcerpt^{XML}.
- Comprehensive and systematic shorthand notations, also for RDF collections.
- Meaningful querying of blank nodes by means of concise bounded descriptions.
- Constructs for expressing negative statements.
- An intuitive semantics based on the simulation unification of Xcerpt^{XML}, the core technique making incomplete queries possible.

This thesis shall show that simulation unification turns out to be very useful for querying RDF as it is appropriate to evaluate queries on the data model of RDF.

Chapter 1 is this introduction. Chapter 2 introduces state-of-the-art serialization formats and query languages which serve as models for the design of Xcerpt^{RDF}. Chapter 3 introduces the syntax of Xcerpt^{RDF} and explains the language constructs with numerous examples. Chapter 4 provides a formal description of normalizing Xcerpt^{RDF} terms simplifying the introduction of the semantics of Xcerpt^{RDF} which is given in Chapter 5. The thesis finishes with a conclusion and an outline.

Chapter 2

Preliminaries

This chapter gives a short and concise overview of RDF constructs relevant for this thesis and also introduces important modelling and query languages for RDF serving as models for the design of Xcerpt^{RDF}.

2.1 The Data Model of RDF

The Resource Description Framework [4, 9, 17, 15, 19] is a Semantic Web formalism in which entities (*resources* or *labels*) are related to another by binary relations (*predicates* or *properties*). There are three kinds of resources: URIs [6], literals and blank nodes. URIs are globally visible and, hence, also addressable from other RDF graphs. Furthermore, URIs are considered to uniquely identify resources, i.e. each occurrence of the same URI denotes the same resource. In accordance to [19], URIs may also be abbreviated by qualified names. For example, the URI http://www.example.org/persons#john may be abbreviated by eg:john, where the prefix eg stands for the namespace http://www.example.org/persons#. In this thesis, URIs are abbreviated by qualified names whenever possible. For frequently used namespaces, this thesis commits to the prefixes given in Table 2.1.

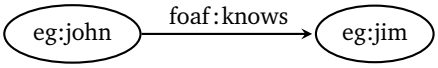
Prefix	**Namespace**	**Comment**
rdf	http://www.w3.org/1999/02/22−rdf−syntax−ns#	The namespace for RDF built-in vocabulary [4]
rdfs	http://www.w3.org/2000/01/rdf−schema#	namespace for RDF Schema vocabulary [9]
foaf	http://xmlns.com/foaf/0.1/	namespace for the FOAF vocabulary [10]
eg	http://www.example.org/xcerptrdf#	namespace for example resources

Table 2.1: Prefixes and the namespaces for which they stand throughout this thesis

Literals are constant values which are represented as character strings. Literals can be typed by or can be tagged by XML Language Tags [20]. Blank Nodes represent anonymous (unnamed) resources and can be seen as existential quantified variables.

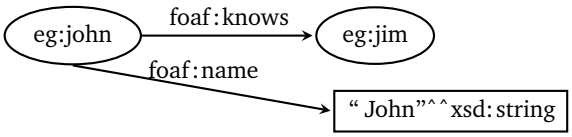
Statements in RDF are called *triples*, each consisting of a subject label, a predicate or property label and an object label. Furthermore, RDF does not distinguish between URI occurrences, except for property resources. This leads to graph-structured data with directed and labelled arcs (the properties) each connecting two labels (subject and object). Labels are also usually called nodes because of the graph structured data model.

For example, the following RDF graph states that a person John knows another person Jim, whereby John is represented by the URI eg:john, the relation knows by the URI foaf:knows and Jim is represented by the URI eg:jim.



Triples can be seen as logical formulas. The triple above can be written as the logical formula foaf:knows(eg:john, eg:jim), where foaf:knows is a relation symbol, eg:john and eg:jim being constants.

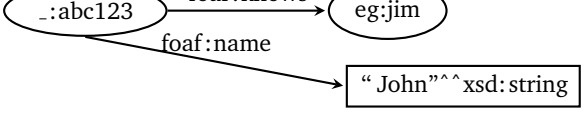
Using blank nodes, the following graph expresses that there exists someone who knows jim (whereby knows is further on represented by foaf:knows and Jim by eg:jim) and whose name is John (whereby the name-relation is represented as foaf:knows).



The use of the blank node in the graph above aims at conveying that the person who knows Jim and who is named John is not fully identified.

Note that the literal together with its type which, by convention, is attached to the character string by a preceding “^^” is a single node in the graph. The blank node is represented as an empty node in the graph. This notation is in line with [19].

However, for RDF modelling languages it is necessary to distinguish between blank nodes. Otherwise, it is not possible to express graph-structured relationships. Hence, blank nodes can be given names which, however, are only declared in the scope of the graph in which they occur. As their names are not visible outside the graph, one says that blank nodes do not have an intrinsic name. Blank nodes are named by so-called “blank node identifiers”. Resembling the notation of [19], blank node identifiers are written as “_:” followed by an identifier. Using blank node identifiers, the latter example is as follows.



There, _:abc123 represents the blank node within the graph. Note that in analogy to URIs blank nodes with identical blank node identifiers are supposed to be identical, whereas blank nodes with different blank node identifiers do not necessarily represent different resources.

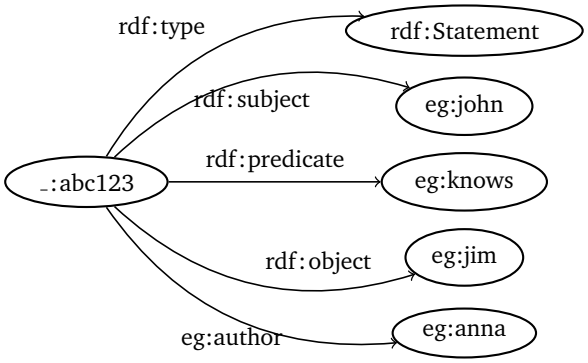
During reasoning, blank nodes might be identified with other nodes (blank nodes or non-blank nodes).

2.2 Built-In Vocabularies of RDF

As described in [19] particular sets of URIs which are used for special purposes are referred to as *vocabularies*. These built-in properties share the namespaces http://www.w3.org/1999/02/22−rdf−syntax−ns# which is usually abbreviated by the prefix rdf. For example, the property rdf:type is used to specify the concrete type of a resource.

2.2.1 Reification

RDF provides reification as means to make statements about statements. This is useful when additional information (so-called provenance information) describes existing statements. Such information comprises, amongst other things, the date of asserting the statement and the author of the statement. Thus, it is possible to state that there exists a statement which expresses that the person John knows the person Jim and which is asserted by (represented as eg:author) the person Anna as pictured by the following RDF graph.



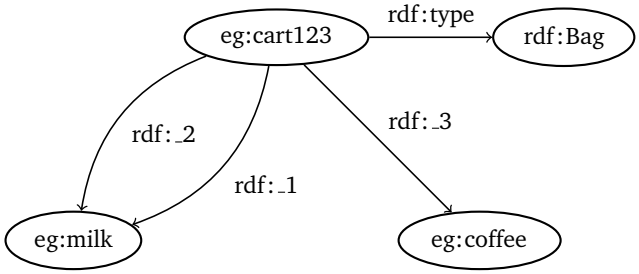
RDF therefore offers the built-in resources rdf:Statement which specifies that a statement is a reification of another statement, rdf:subject, rdf:predicate and rdf:object for the specification of the reified statement itself.

2.2.2 Containers and Collections

RDF offers built-in vocabularies for describing container and collection datatypes such as bags (multisets), sequences (vectors), alternatives and lists. A tutorial explanation of containers and collections is given in [19]. The kind of the datatype is given by the rdf:type declaration, cf. following possibilities.

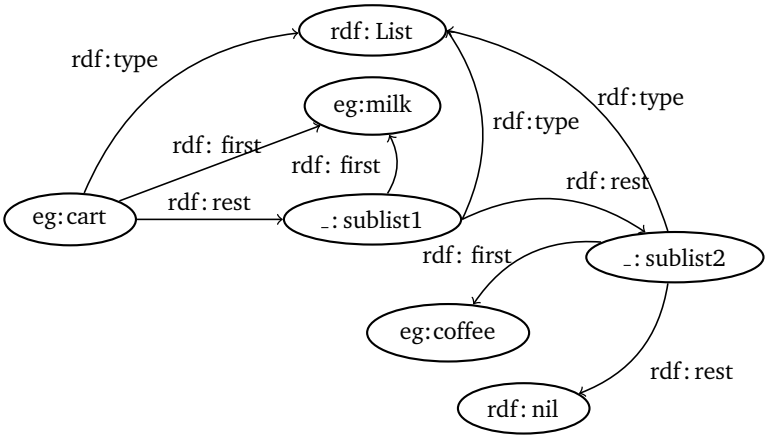
Container	**Type Declaration**
bag (multiset)	rdf:Bag
sequence (vector)	rdf:Seq
alternative	rdf:Alt
Collection	**Type Declaration**
list	rdf:List

The elements of an RDF container are declared by so-called *membership-properties* which are of the form rdf:_N where N stands for a numeric literal. For example, rdf:_1, rdf:_13 are membership-properties, whereas rdf:_12f is not. The following RDF graph models a shopping cart (represented as eg:cart123) as an rdf:Bag container which contains one item of eg:coffee and two items of eg:milk.



RDF containers are considered unclosed, i.e. in RDF it is not possible to express that an RDF container exclusively comprises the declared members. For rdf:Bag and rdf:Alt the order of the numbering of the membership-properties (rdf:_1, rdf:_2, and further) are irrelevant. However, the numbering is relevant to distinguish between multiple occurrences of identical elements within the container.

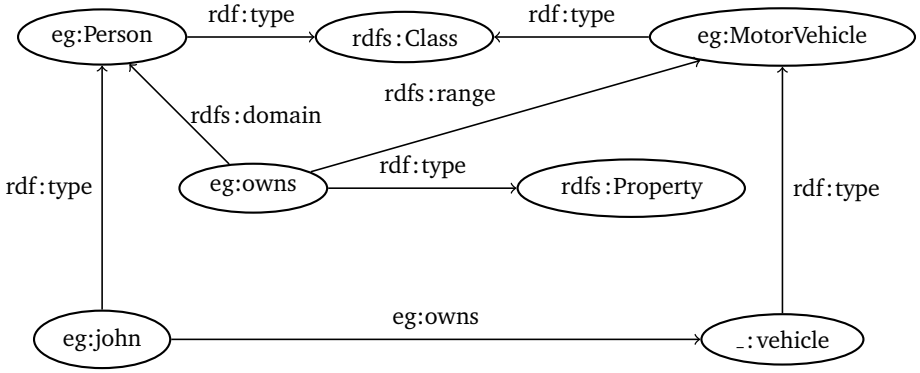
In contrast to RDF containers, lists as RDF collections can be closed – the empty list is described by the resource rdf:nil. The structure of a list is straightforward: A list consists of a list head (declared by the property rdf:first) and a tail (declared by the property rdf:rest) which in turn is a list. The following graph models the shopping cart of the latter example as the list [milk,milk,coffee].



2.2.3 RDF Schema

RDF Schema [9] is a special vocabulary with which one can define application specific vocabularies. This comprises the description of new resource classes and new properties. New classes are defined by asserting that the rdf:type of a resource is rdfs:Class. Properties are described by asserting that the rdf:type of a resource is rdfs:Property. Furthermore, the domain (i.e. the set of potential subjects of the property) can be described by rdfs:domain and the range (i.e. the set of potential objects of the property) by rdfs:range.

The following graph states that eg:MotorVehicle and eg:Person describe classes, and eg:owns is a property that relates persons to their vehicles. Furthermore, it states that eg:john is a person who owns a vehicle of type eg:MotorVehicle.



In RDF classes and properties are integrated in a class (property, resp.) hierarchy, each class is a subclass of rdfs:Resource and each property is a subclass of rdfs:Property. In order to describe that a class is a sub-class of another class, the property rdfs:subClassOf is used. In order to describe that a property is a sub-property of another, the property rdfs:subPropertyOf is used.

2.3 Introduction to Turtle, SPARQL and TRIPLE

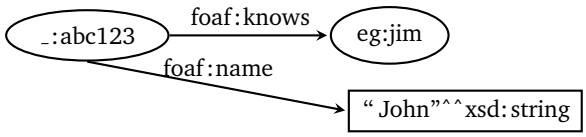
This section gives a short overview of the well-known and standard RDF serialization syntax Turtle and the RDF query languages SPARQL and TRIPLE. The introduction focuses more on the syntax than on the query evaluation. Several papers discuss and compare the evaluation of queries of the different RDF query languages.

2.3.1 Turtle - Terse RDF Triple Language

Turtle [3] is a concrete syntax for RDF. It is a subset of N3 [5] and, thus, is generally usable in systems that support N3. N3, Turtle and derivatives are very easy to use, which certainly is the reason for several standard query languages being based on these syntaxes.

The building blocks of Turtle are triples of the form <SUBJECT> <PREDICATE> <OBJECT>.. The period finishes the triple. <SUBJECT> and <OBJECT> may be URIs, QNames or Blank

Nodes. <PREDICATE> may be URIs or QNames. In order to simplify parsing, URIs are distinguished from QNames by parenthesizing them in angle brackets. Blank Nodes are written as _:<IDENT>, where <IDENT> is a valid identifier. Turtle offers the keyword @prefix to declare namespace prefixes. The following RDF graph can be described in Turtle as follows.




```
@prefix eg:   <http://www.example.org/>
@prefix foaf: <http://xmlns.com/foaf/0.1/>
@prefix rdf:  <http://http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>

_:abc123 foaf:knows eg:john.
_:abc123 foaf:name  "John"^^xsd:string.
```

Turtle offers shorthand notations for abbreviating groups of triples, sometimes referred to as factorization. Therefore, the comma is used to repeat the subject and predicate of triples only differing in the objects (subject-predicate factorization):

```
eg:a eg:p eg:o1,
          eg:o2,
          eg:o3.
```

This excerpt is syntactic sugar for the following triples:

```
eg:a eg:p eg:o1.
eg:a eg:p eg:o2.
eg:a eg:p eg:o3.
```

Additionally, the semi-colon is used to repeat the subject of triples which only differ in the predicates and objects. The excerpt

```
eg:a eg:p1 eg:o1;
     eg:p2 eg:o2;
     eg:p3 eg:o3.
```

abbreviates the following triples:

```
eg:a eg:p1 eg:o1.
eg:a eg:p2 eg:o2.
eg:a eg:p3 eg:o3.
```

Finally, Turtle offers shorthand notations for RDF collections (RDF lists). A collection is written as a parenthesized list of labels. The description

```
1 eg:a eg:p (eg:i1, eg:i2, eg:i3).
```

stands for the following triples:

```
1  eg:a eg:p      _:b1.
2
3  _:b1 rdf:first eg:i1;
4       rdf:rest  _:b2.
5
6  _:b2 rdf:first eg:i2;
7       rdf:rest  _:b3.
8
9  _:b3 rdf:first eg:i3;
10      rdf:rest  rdf:nil.
```

Note that this shorthand notation implicitly expresses blank nodes, i.e. the expansion of this shorthand notation yields blank nodes. In particular, (eg:a, eg:b) describes a valid graph in Turtle since it can be expanded to the following triples.

```
1 _:b1 rdf:first eg:a;
2      rdf:rest  _:b2.
3 _:b2 rdf:first eg:b;
4      rdf:rest  rdf:nil.
```

2.3.2 SPARQL

SPARQL [21] is a query language for RDF which has recently reached W3C Recommendation status. It is based on a non-XML syntax comparable to Turtle. Additionally, SPARQL offers the following further language constructs.

2.3.2.1 Anonymous Blank Nodes

Beside the named blank nodes, SPARQL offers anonymous blank nodes (wildcards) which are written as [], i.e. square brackets. Each occurrence of [] denotes a new blank node not occurring elsewhere in the document. Using anonymous blank nodes, SPARQL offers two forms of writing the same triple.

```
1 [] eg:p eg:o.
2 [eg:p eg:o].
```

Within these square brackets, Turtle factorizations may occur as shown in line 2. This allows to describe several triples sharing the same anonymous blank node as subject. Note that

```
1 [] eg:p1 eg:o1;
2    eg:p2 eg:o2.
```

is not the same as

```
[ eg:p1 eg:o1;
  eg:p2 eg:o2 ].
```

since the first triples refer to two distinct blank nodes whereas the second triples refer to the same anonymous blank node.

2.3.2.2 Nested Collection Elements

Within the shorthand notation for RDF collections as introduced in Turtle, SPARQL allows that certain syntactic sugar for triples may occur as collection elements. This comprises either RDF collections themselves or triples within square brackets (i.e. referring to anonymous blank nodes). The following description shows composition (taken from [21]).

```
(1 [:p :q] ( 2 ) ) .
```

The collection above can be expanded as follows.

```
_:b0  rdf:first  1 ;
      rdf:rest   _:b1 .
_:b1  rdf:first  _:b2 .
_:b2  :p         :q .
_:b1  rdf:rest   _:b3 .
_:b3  rdf:first  _:b4 .
_:b4  rdf:first  2 ;
      rdf:rest   rdf:nil .
_:b3  rdf:rest   rdf:nil .
```

2.3.2.3 SPARQL Query Constructs

SPARQL offers several ways to query data. Using SELECT-queries only variable bindings are returned (just like in SQL), whereas CONSTRUCT-queries yield RDF graphs. That is, SPARQL supports answer-closedness. ASK-queries only return "yes/no" answers. Neither RDF graphs, nor variable bindings are computed. Finally, SPARQL supports the notion of so-called concise bounded descriptions by DESCRIBE-queries.

SPARQL supports the querying of optional values, but it does not provide syntax for negation. Negation is realized in SPARQL by a workaround using the predicate isBound by which it can be tested whether variables are bound to values or not.

The following program queries the URIs which represents the books and the title of the books the URIs refer to. The RDF document is assumed to be retrievable at http://www.example.org/books.rdf.

```
PREFIX dc: <http://purl.org/cd/elements/1.1/>
PREFIX ex: <http://example.org/book/>
SELECT ?book, ?title
FROM <http://www.example.org/books.rdf>
WHERE { ?book dc:title ?title}
```

The program below queries the same as the program above but yields an RDF graph instead of just variable bindings.

```
PREFIX dc: <http://purl.org/cd/elements/1.1/>
PREFIX ex: <http://example.org/book/>
PREFIX eg: <http://www.example.org#>
CONSTRUCT { ?book eg:hasTitle ?title }
FROM <http://www.example.org/books.rdf>
WHERE { ?book dc:title ?title}
```

2.3.3 TRIPLE

TRIPLE [23] is a query, inference and transformation language for RDF. It allows flexible abbreviations not only applicable for the declaration of namespace prefixes but also for replacing entire QNames being frequently used. For example, the declaration isa := rdf:type allows to use the name isa as a replacement of rdf:type.

TRIPLE provides a slot-oriented syntax i.e. a Turtle triple eg:s eg:p eg:o. is written in TRIPLE as eg:s[eg:p -> eg:o]. Furthermore, TRIPLE supports molecules as factorizations of subjects. The Turtle triples

```
eg:s eg:p1 eg:o1;
     eg:p2 eg:o2.
```

correspond to the TRIPLE molecule eg:s[eg:p1 -> eg:o1; eg:p2 -> eg:o2]. The slotted notation also allows composition. For example, the Turtle triples

```
eg:s  eg:p1 eg:o1.
eg:o1 eg:p2 eg:o2.
```

can be written in TRIPLE as eg:s[eg:p1 -> eg:o1[eg:p2 -> eg:o2]].

Remarkably, TRIPLE does not support explicit blank nodes and neither does it provide shorthand notations for RDF collections. Rather, TRIPLE provides syntactic sugar for reification which implicitly expresses a blank nodes. The turtle triples

```
eg:Anna  eg:believes    _:b1.
_:b1     rdf:type       rdf:Statement;
         rdf:subject    eg:John;
         rdf:predicate  foaf:knows;
         rdf:object     eg:Jim.
```

can be described in TRIPLE as eg:anna[eg:believes -> <eg:john[foaf:knows -> eg:Jim]>].

2.4 Introduction to Xcerpt^{XML}

As this thesis deals with the extension of Xcerpt^{XML}, a short introduction of this language is also given. Xcerpt^{XML} [22] is a deductive, pattern based query and transformation language for semi-structured data. Xcerpt^{XML} uses terms as patterns for representing, querying and constructing semi-structured data. These kinds of data are described in the following sections.

2.4.1 Data Terms

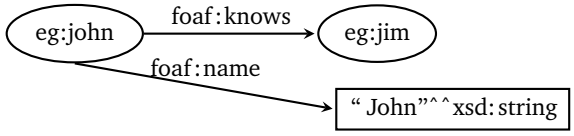
Data terms are abstract term representations of semi-structured data in Xcerpt^{XML}. For a given element of the form <ns:elem>...</ns:elem> the term representation is either ns:elem{...} or ns:elem[...]. The curly braces specify that the children of the element ns:elem (also called immediate subterms of ns:elem) are considered as unordered. The square brackets specify that the immediate subterms are considered as ordered. For the translation of XML to Xcerpt^{XML} data terms, the subterms are considered as ordered by default. XML attributes of an element are represented as subterms of the element. That is, an XML element

```
<ns:elem attr1=value1 attr2=value2 ... attrN=valueN>
...
</ns:elem>
```

is represented in Xcerpt^{XML} as follows.

```
ns:elem [
 attributes{ attr1[ value1 ],
             attr2[ value2 ],
             ...,
             attrN[ valueN ]
 }
 ...
]
```

To sum up the introduction of Xcerpt^{XML} data terms, the RDF/XML representation of the following RDF graph is given as Xcerpt^{XML} data term.



```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf  ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
         xmlns:eg   ="http://www.example.org/"
         xmlns:foaf ="http://xmlns.com/foaf/0.1/">

  <eg:john>
    <foaf:knows><eg:jim /></foaf:knows>
    <foaf:name rdf:datatype="http://www.w3.org/2001/XMLSchema#string">
      John
    </foaf:name>
  </eg:john>

</rdf:RDF>
```

The Xcerpt^{XML} data term representing the XML document above is as follows. Note the declaration of namespaces with the keyword ns−prefix as shown below.

```
ns-prefix rdf  = "http://www.w3.org/1999/02/22-rdf-syntax-ns#"
ns-prefix eg   = "http://www.example.org/"
ns-prefix foaf = "http://xmlns.com/foaf/0.1/"

rdf:RDF [
  eg:john [
    foaf:knows[ eg:jim[ ] ],
    foaf:name [
      attributes{
        rdf:datatype["http://www.w3.org/2001/XMLSchema#string"]
      },
      "John"
    ]
  ]
]
```

2.4.2 Query Terms

Query terms are patterns which usually contain variables (introduced by the keyword var). They are matched against data to determine variable bindings, thus, the resulting ground query term can be found in the data. Thereby, amongst others, it is possible to specify the following conditions, which must be fulfilled.

Incompleteness in Breadth The data term the query term is matched against must contain at least the subterms specified by the query term but may contain further subterms.

Incompleteness in Depth The query is matched against data which is at arbitrary depth.

Optional Subterms Subterms within the query term may be specified as optional so that matching the query term against data terms not providing suitable subterms for the optional subterms is considered to be successful.

Subterm Specification of the Query Term	Condition for Matching Data Terms by the Query Term
$a[s1, s2]$	Complete and ordered: there may be only the subterms in the order $s1, s2$
$a\{s1, s2\}$	Complete and unordered: there may be only the subterms $s1$ and $s2$ but they may also occur in the order $s2, s1$.
$a[[s1, s2]]$	Incomplete and ordered: there may be more than the subterms $s1$ and $s2$ but $s1$ must immediately precede $s2$.
$a\{\{s1, s2\}\}$	Incomplete and unordered: there may more than the subterms $s1$ and $s2$. It is just required that $s1$ and $s2$ are also available.

Table 2.2: Subterm specifications of query terms

Negation of Subterms Subterms may be specified as negative such that within suitable data terms there are no subterms against which these query subterms can be matched.

Incompleteness in breadth is specified using double braces. In conjunction with the specification of the order of subterms, four kinds of bracketing can be used in query terms. These are listed in Table 2.2.

Incompleteness in depth is specified by the keyword desc and specifies that the query term following desc may match with all data terms that contains subterms at arbitrary depth that is matched by the query term.

Query terms containing optional subterms specified by the keyword optional also match against data terms which do not contain subterms which can be matched by the optional subterms. In the case of optional subterms not matching against the data subterms, the query still succeeds but does not yield bindings for the variables occurring in the optional subterm, i.e. optional subterms only affect variable bindings.

Query terms containing negative subterms which are specified by the application of without only match with data terms that do not contain subterms which are matched by negative subterms. Note that negative subterms do not yield any bindings. Thus, all variables occurring within negative subterms must already be bound. This is the case when variables within a negative subterm also occur within a positive subterm.

For example, consider the following query term.

```
1 var U {{
2   var V{{"1"}}
3   optional var W{{ var X{{"10"}} }}
4   without  var V{{"3"}}
5 }}
```

This query term matches against the data terms a[b["1"]], a [b["2", "1"], c[d["10"]]] but not against a[b["2"]] and neither against a[b["1"], b["3"]].

In Xcerpt^{XML}, queries may be query terms or recursively built by queries using the binary connectives and{t,t'}, or{t,t'} or the unary connective not t.

Input resources can be assigned to queries. For example, suppose the query above shall be matched against data which can be found at http://www.example.org/test.xml. Then the query looks as follows:

```
1 in {
2   resource[http://www.example.org/test.xml,]
3   var U {{
4     var V{{"1"}}
5     optional var W{{ var X{{"10"}} }}
6     without  var V{{"3"}}
7   }}
8 }
```

2.4.3 Construct Terms

Construct terms consume variable bindings resulting from the matching of queries against data in order to create new data. That is, construct terms make Xcerpt^{XML} answer-closed.

Construct terms are similar to data terms, however, variables and the grouping constructs all and some may occur. Since construct terms consume variable bindings resulting from the evaluation of query terms the variables within the construct terms must be a subset of the variables within the query terms. This is referred to as range restrictedness.

For each variable binding the construct term creates a new instance of the answer term. However, it is often desired that for some or all bindings of certain variables only one answer term is created. That is, the answer term shall group the variable bindings according to the values of certain variables. This is done by using the grouping constructs all and some n. For example, consider the following bindings.

Bindings	**Value**
var U	person[name["Anna"]]
var V	person[name["John"]]
var U	person[name["Anna"]]
var V	person[name["Jim"]]

The construct term friends[knows[var U, var V]] creates the following answer terms:

```
friends [
 knows [
  person[ name["Anna"] ],
  person[ name["John"] ]
 ]
]

friends [
 knows [
  person[ name["Anna"] ],
  person[ name["Jim"] ]
 ]
]
```

The construct term friends[all knows[var U, var V]], however, yields the following answer:

```
friends [
 knows [
  person[ name["Anna"] ],
  person[ name["John"] ]
 ]

 knows [
  person[ name["Anna"] ],
  person[ name["Jim"] ]
 ]
]
```

If the answer shall be grouped according to variables which do not appear within the answer, the explicit grouping applied by the construct group by {<VARIABLES>} can be used. For example, the construct term

```
friends[ all knows[ var V ] ] group by {var U}
```

yields the following answer:

```
friends [
 knows [
  person[ name["John"] ]
 ]
 knows [
  person[ name["Jim"] ]
 ]
]
```

2.4.4 Xcerpt^{XML} Programs

Simple Xcerpt^{XML} programs consists of rules (goal and construct) and hence have the following form.

```
1  GOAL
2    out {
3      resource{<RESOURCE>},
4      <CONSTRUCT-TERM>
5    }
6  FROM
7    <QUERY>
8  END
9
10 CONSTRUCT
11   <CONSTRUCT-TERM>
12 FROM
13   <QUERY>
14 END
15
16  ...
17
18 CONSTRUCT
19   <CONSTRUCT-TERM>
20 FROM
21   <QUERY>
22 END
```

Each Xcerpt^{XML} program has at least one goal. Goals may be associated with output resources to which the answer (i.e. the instantiations of answers by the construct term of the goal) is written. If the output resource is not specified the answer is written to stdout. Construct rules associate construct terms to query terms. A program may comprise an arbitrary amount of construct rules (including zero).

Chapter 3

Xcerpt^{RDF} Language Constructs

In this chapter the syntax of Xcerpt^{RDF} for querying and delivering RDF data is introduced. The syntax is basically inspired by Triple [23] which, in turn, is inspired by F-Logic [16]. Hence, Xcerpt^{RDF} ,like Triple and F-Logic, supports two different views on the data which is valuable for the modelling. On the one hand, the syntax captures the view of RDF data as directed graphs. On the other hand, it captures the view of RDF data as objects (in the sense of object-oriented modelling), their attributes and their values. How both views are supported by Xcerpt^{RDF} is shown in this chapter. Furthermore, the additional syntax constructs of Xcerpt^{RDF} integrates well with that of Xcerpt^{XML} for reasons which are also discussed in this chapter. Furthermore, Xcerpt^{RDF} offers various shorthand notations to simplify the authoring of RDF facts and queries. These shorthand notations comprise constructs for frequently used subgraph-structures of RDF as well as syntactic means to compose RDF data coherently and with few redundancies.

3.1 RDF Terms in Xcerpt^{RDF}

As mentioned above, every RDF triple consists of a subject node which can either be a URI or a blank node, a property which can only be a URI and an object which can either be a URI, a blank node or a literal. In this section, it is shown how these RDF terms (i.e. URIs, blank nodes and literals) are represented in Xcerpt^{RDF}. In many cases their notations adhere to the conventions that have been established by N3 and other RDF serialization formats [7].

Xcerpt^{RDF} offers several ways to represent resources. The simplest and certainly most frequent way is to use qualified names [25]. For instance, the person called John could then be represented as eg:John, where the prefix eg represents the namespace http://www.example .org/. Prefixes in Xcerpt^{RDF} are declared the same way as it is done in Xcerpt^{XML}, i.e. by ns−prefix eg = http://www.example.org/. Another way to represent the person John would be to write the entire URI, i.e. http://www.example.org/John. In order to simplify parsing (i.e. to differentiate between qualified names and URIs), one can also think of prefixing the URI with @, so that the notation of the URI would be @http://www.example.org/John and the notation of the respective qualified name eg:John.

While the usage of blank nodes is clear, the ways of their expression in query languages is

diverse. They can either be expressed *explicitly*, i.e. by the user, or *implicitly*, i.e. by the system. Explicit blank nodes can be subdivided into those that are *named explicitly* by the user or *named implicitly* by the system. The syntax of explicitly named blank nodes corresponds to the syntax which is introduced above, i.e. _:abc123, where _: specifies that the node which is identified as abc123 is a blank node. As in other RDF query languages like SPARQL, when a user wants the system to name a blank node, he simply omits the identifier. Hence, an implicitly named blank node is written as _:. For each occurrence of _:, Xcerpt^{RDF} creates a new and unique identifier which, hence, is not addressable from elsewhere. The implicit naming of blank nodes is, thus, reminiscent of the naming of anonymous variables or wildcards, e.g. as they are known from e.g. Prolog.

Reserving the underscore as the namespace for blank nodes leads to conflicts, since the underscore is a possible namespace prefix in XML and, thus, free. These conflicts could be resolved by transforming the original document to another, thereby, replacing the underscore by another conflict-free namespace prefix. Another possibility is to disallow the usage of the underscore as XML namespace prefix when Xcerpt^{XML} is used so that it may not be used for purposes other than defining blank nodes. This makes sense as the underscore is supposed to be seldom used as a user-defined namespace prefix.

Implicit blank nodes are created by the system during the expansion of shorthand notations which are offered in many RDF query languages, e.g. Triple [23], SPARQL [21] and SeRQL [11] for extensive and frequently used constructs such as reification or collections. These shorthand notations express blank nodes without any assistance by the user. As a consequence, the user can only take advantage from these shorthand notations when blank nodes that cannot be addressed from elsewhere in the RDF graph are sufficient. Otherwise the corresponding longhand notations (i.e. the corresponding RDF triples) have to be used instead. This problem is overcome in Xcerpt^{RDF} by more flexible shorthand notations for reification, collections and containers so that Xcerpt^{RDF} can get along without implicit blank nodes.

Literals are written in Xcerpt^{RDF} the same way as it is the case in N3, SPARQL and others, i.e. the syntax of literals are quoted strings like "abc". Language tags are separated from the literal string by the symbol @, for example "abc"@de. Language tags are optional. Moreover, optional type URI for literals are introduced in Xcerpt^{RDF} with ^^, for example "abc"^^xsd:string, as it is also done in most other RDF query languages.

Using the symbol @ for separation of the language tag from the literal does not lead to confusions with its usage for references in Xcerpt^{XML} and either for the differentiation between URIs and qualified names since references are not used in Xcerpt^{RDF} and language tags are preceded by strings.

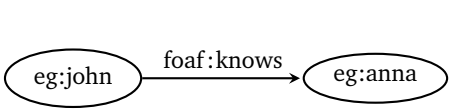
3.2 Data Terms

Data terms are abstract representations of RDF data in Xcerpt^{RDF}. They also serve as a serialization format for RDF graphs. Thus, they do not contain (Xcerpt^{RDF}) variables. Note that in Xcerpt^{RDF} one distinguishes between blank nodes and variables so that also RDF statements containing blank nodes are considered ground. For frequently used RDF constructs Xcerpt^{RDF}

offers several shorthand notations which simplify the authoring and manipulation of RDF data. These shorthand notations are introduced in this section.

3.2.1 Simple Statements

Simple statements in Xcerpt^{RDF} correspond to RDF triples and hence consist of a subject, property and an object. The following example shows an RDF triple as a graph and the corresponding notation in Xcerpt^{RDF}.




```
1 eg:john {
2   foaf:knows → eg:anna
3 }
```

Example 3.1: RDF Triple as a graph and as an Xcerpt^{RDF} data term

In Xcerpt^{RDF}, the subject is separated from its subterm, i.e. its property and object by curly braces. The property points to the object by an arrow[1].

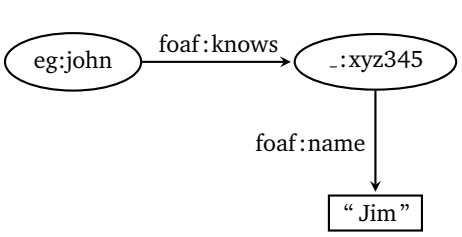
As mentioned above, this syntax supports two different viewpoints, the graph-based and the object-oriented viewpoint. Considering the graph-based viewpoint, the Xcerpt^{RDF} data term immediately corresponds to the directed RDF graph that is shown above: The property together with the arrow represents the edge that connects the subject with the object. This allows for an intuitive authoring of RDF data. When considering the object-oriented viewpoint, in the example above eg:john is the object (in the object-oriented sense) which has as an attribute foaf:knows referencing the object (in the object-oriented sense) eg:anna. Furthermore, this notation resembles the slot notation from frame languages with RDF properties as slots and RDF objects as fillers.

RDF graphs consisting of more than one triple can be described by several Xcerpt^{RDF} data terms which are optionally separated by commas. Commas for separating data terms are optional due to the fact that they are also optional in Xcerpt^{XML}. As a digression, the use of delimiters to revoke injectivity of subterm mappings in Xcerpt^{XML} is discussed in Section 3.8. Example 3.2 shows an RDF graph and its corresponding description in Xcerpt^{RDF} asserting that the person who is represented by the URI eg:john knows someone whose name is Jim.

3.2.2 Composition of Data Terms

In the latter example, the RDF graph is described in Xcerpt^{RDF} with two data terms which correspond to triples. Such a description, however, does not take the structure of the graph into account. That is, the structure of the graph remains implicit. In Xcerpt^{RDF} syntactic sugar is offered in order to describe graphs coherently, thus, making their structures explicit. This syntactic sugar is usually referred to as *composition*. Using composition, the latter example looks

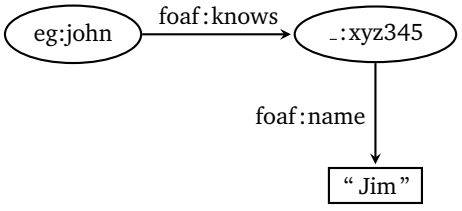
[1] The actual textual representation of this arrow is "-->" (cf. Section 3.3.5).




```
1 eg:john {
2   foaf:knows → _:xyz345
3 }
4
5 _:xyz345 {
6   foaf:name → "Jim"
7 }
```

Example 3.2: An RDF graph and its representation in Xcerpt^{RDF}

as follows.




```
1 eg:john {
2   foaf:knows → _:xyz345 {
3     foaf:name → "Jim"
4   }
5 }
```

Example 3.3: The graph from Example 3.2 and the corresponding composed data term

Data terms can be composed recursively at any depth, i.e. composites of a data term can in turn be composed. It should be mentioned that in composed data terms objects are further on RDF terms and not Xcerpt^{RDF} data terms. The fact that data terms occur at object positions might give the wrong impression that entire data terms may also be object nodes. In Example 3.3, the data term _:xyz345{ foaf:name → "Jim" } occurs at object position, but the actual RDF object retains to be _:xyz345. Composition of data terms is, thus, similar to subordinate sentences. Example 3.3 can be expressed in English using a subordinate sentence such as: "John knows someone who is named Jim".

3.3 Shorthand Notations for Graph Patterns

RDF data can be completely expressed in Xcerpt^{RDF} as it is specified so far. However, several graph structures occur frequently in RDF data. For these, query languages like SPARQL or Triple offer several shorthand notations which are, however, often restricted to certain settings (cf. implicit blank node generation as mentioned in Section 3.1). Other shorthand notations are not offered since they are supposed to be seldom used. This is contrary to a systematic approach. Abandoning systematic language constructs makes program development error prone because the capabilities of stepwise changes and refinements of programs are limited.

In the design of Xcerpt^{RDF} it is strived for a clear, i.e. systematic language which allows for flexible shorthand notations. Besides the abbreviation of extensive but frequently used constructs, shorthand notations can also be used for abstraction, thus placing a special emphasis

on the ideas behind the constructs they abbreviate. For example, in analogy to mathematics, it makes sense to consider the specification of the domain and the range of an RDF property as part of its type rather than just stating that the property is a rdf:Property. Although the specifications of the type, domain and range are independent, the use of a shorthand notation makes their logical interconnections explicit (the shorthand notation for type specification is discussed in Section 3.3.2). Both purposes of shorthand notations are taken into account in Xcerpt^{RDF}.

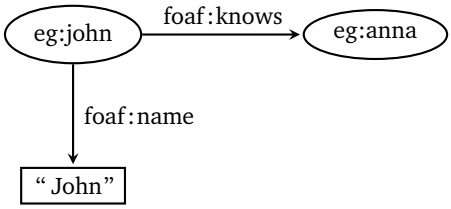
It has been decided that shorthand notations, except factorizations, stand for sets of property object-pairs. This decision not only provides for subject nodes to remain explicit which has advantages especially when considering shorthand notations for reification (cf. Section 3.3.3), collections and containers (cf. Section 3.3.4). It also makes these shorthand notations only occur as subterms, thus, offering the programmer uniformity in their application. Furthermore, this decision admits an in situ expansion of these shorthand notation which simplifies their processing.

3.3.1 Factorizations of Data Terms

Xcerpt^{RDF} offers ways to merge several data terms to a single data term by factoring out identical parts. Whereas other query languages only support certain factorizations, Xcerpt^{RDF} supports all combinations. In factoring out, so-called lists of RDF terms and combinations of them are generated. These lists are usually referred to as subject-lists, subject-property-lists and so on. Lists of RDF terms and also data terms (which is due to the ability of composing data terms) are thereby written as parenthesized sequences whose elements are separated by commas. This is illustrated by several examples below.

3.3.1.1 Factorization of Subjects

The factorization of subjects merges data terms that share the same subject. Therefore, the properties and objects can be arbitrary. Thus, factoring out subjects reduces the expense of writing graph patterns that express 1-to-n relationships between the subject and its objects. This kind of factorization occurs often and is therefore also offered by several RDF serialization formats such as Turtle (cf. Section 2.3.1). The factorization of subjects results in lists of property-object pairs. This is illustrated in the following example:




```
1 eg:john {
2   foaf:knows → eg:anna,
3   foaf:name  → "John"
4 }
```

Example 3.4: The Xcerpt^{RDF} data term using factorization of subjects

The above code fragment in Example 3.4 is equivalent to the following two data terms.

```
1 eg:john { foaf:knows → eg:anna }
2 eg:john { foaf:name  → "John" }
```

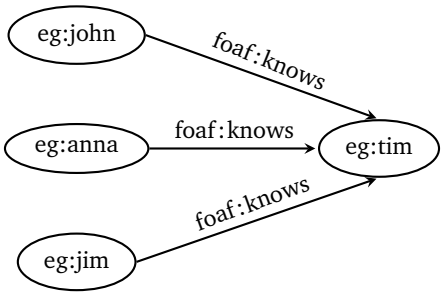
Lists of property-object pairs are written between the curly braces of the data term. This notation supports the object-oriented viewpoint which is mentioned above. The data term in the example above can be seen as an object-specification, i.e. as an instance with its attributes and their values. Predicate-object pairs may be separated by commas. Such a separation can be useful as points for the parser to continue its work in cases of syntactic errors. However, from the modelling point of view these commas are not necessary and, hence, can be omitted.

Now that this shorthand notation is introduced, the use of curly braces for separating the subject from its subterms makes sense: in Xcerpt^{XML}, curly braces specify that the order of subterms is irrelevant. Since in RDF there is no inherent order of (sub-)terms using the same bracketing is consistent to Xcerpt^{XML}.

Though being a shorthand notation, for the issues dealt with in this thesis there is no need to expand this factorization to its corresponding longhand notation.

3.3.1.2 Factorization of Predicate-Object Pairs

Factoring out property-object pairs serves as a shorthand notation of graph patterns which express n-to-1 relationships where each subject is related to the same object by the same property. The factorization of property-object pairs results in subject-lists. Consider the Example 3.5 in which it is asserted that the three persons represented as eg:john, eg:anna, eg:jim know the person represented as eg:tim.




```
1 (eg:john, eg:anna, eg:jim) {
2   foaf:knows → eg:tim
3 }
```

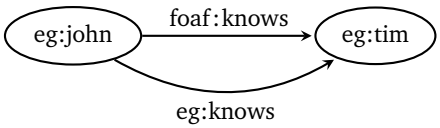
Example 3.5: Xcerpt^{RDF} data term using factorization of property-object pairs

As explained in the introductory part of this section the subject-list consisting of eg:john, eg:anna and eg:jim is written as a parenthesized sequence of RDF terms. The Xcerpt^{RDF} data term of Example 3.5 is a shorthand notation for the following three statements:

```
1 eg:john { foaf:knows → eg:tim }
2 eg:anna { foaf:knows → eg:tim }
3 eg:jim  { foaf:knows → eg:tim }
```

3.3.1.3 Factorization of Subject-Object Pairs

The factorisation of subject-object pairs is not offered in most RDF query languages since corresponding RDF graph structures are considered to rarely occur. However, since it is argued for a systematic language design factoring out subject-object pairs is supported in Xcerpt^{RDF}. The factorisation of subject-object pairs results in property-lists and serves as a shorthand notation for RDF graph structures that express multiple relationships between a subject and an object. Such structures occur in cases when graphs are integrated to a single graph where different vocabularies shall still be available. In Example 3.6 this is illustrated by two persons which know each other but where this "knows"-relationship is expressed in two different vocabularies.




```
1 eg:john {
2  (foaf:knows, eg:knows)
3    → eg:tim
4 }
```

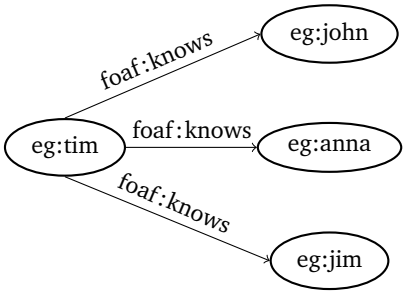
Example 3.6: Xcerpt^{RDF} data term with factorization of subject-object pairs

The RDF graph of Example 3.6 merges the two data terms below.

```
1 eg:john { foaf:knows → eg:tim }
2 eg:john {    eg:knows → eg:tim }
```

3.3.1.4 Factorization of Subject-Predicate Pairs

The factorization of subject-property pairs results in object-lists. It is useful especially for abbreviating data terms which express 1-to-n relationships where the subject is related to n objects by the same property. Consider the Example 3.7 which shows that the person represented by eg:tim knows the persons represented by eg:john, eg:Anna and eg:jim.




```
1 eg:tim {
2   foaf:knows →
3    (eg:john,eg:anna,eg:jim)
4 }
```

Example 3.7: Factorization of subject-property pairs

The RDF graph of Example 3.7 can also be equivalently described by the following data terms.

Factorization	Resulting List	Example

subject	property-object-list	s {
		p1 → d1,
		...
		pn → dn
		}

property	subject-object-list	(s1,...,sn) {
		p → (d2,...,dm)
		}

object	subject-property-list	(s1,...,sn) {
		(p1,...,pm) → d
		}

subject, property	object-list	s { p → (d1,...,dn) }
subject, object	property-list	s { (p1,...,pn) → d }
property, object	subject-list	(s1,...,sn) { p → d }

subject, property, object	subject-,
	property-,
	object-list
		(s1,...,sl) {
		(p1,...,pm) → (d1,...,dn)
		}

Table 3.1: Possible factorizations of data terms and resulting lists

```
1 eg:tim { foaf:knows → eg:john },
2 eg:tim { foaf:knows → eg:anna },
3 eg:tim { foaf:knows → eg:jim  }
```

3.3.1.5 More Factorizations

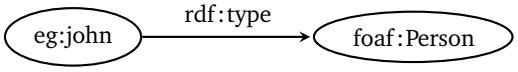
In the previous paragraphs the basic factorizations are introduced. In Xcerpt^{RDF} it is also possible to freely combine these factorizations. Hence, in conjunction with composition it is possible to describe an entire RDF graph with a single data term while making the structure and m-to-n relationships explicit. There are seven different factorizations possible. An overview is given in Table 3.1.

3.3.2 Types for Nodes and Properties

The type description for nodes is a graph pattern which is certainly often used. Although the notation is not extensive (since it consists of only one rdf:type edge), a shorthand notation for it is discussed for the following reasons. First, in this thesis it is aimed at shorthand notations for all frequently used graph patterns. Second, other query and RDF languages such as N3 [5] also offer shorthand notations for rdf:type. Third, this shorthand notation shall provide the

typing of both nodes and properties, whereas for properties it also shall be possible to specify their domains and ranges. This kind typing specification resembles the “typing” of relations in mathematics. Such a shorthand notation is currently not provided by existing query languages (cf. [13], Section 2.3).

The typing of nodes can be abbreviated by the is construct. Consider the Example 3.8 in which it is stated that eg:john is of the type foaf:Person.

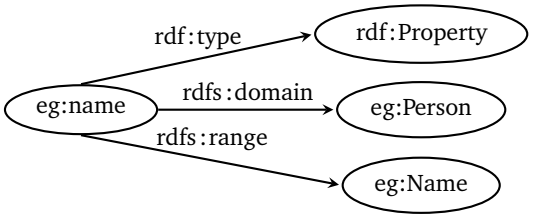



```
1 eg:john {
2   is foaf:Person
3 }
```

Example 3.8: The shorthand notation for specifying the type of a node

Alternatively, it has been suggested to suffix the resource by ^^ followed by the URI of the type in analogy to the typing of literals. However, differences between typed literals and typing of nodes exist. Whereas typed literals are represented as single nodes in RDF, the typing of nodes involves an rdf:type edge and two nodes, the subject and the type URI. This might lead to confusion and, thus, argues against the usage of the same construct for different notions of typing.

Furthermore, the shorthand notation for typing nodes can be extended for properties. For typing properties, the is construct is extended by two URIs specifying the domain and range within square brackets. The domain is separated from the range by an arrow[2]. These square brackets indicate that the URIs are considered ordered since the first URI specifies the domain and the second specifies the range. Additionally, the parsing is easier since the opening square brackets clearly distinguishes the typing of properties between the typing of nodes. Moreover, the brackets also support a clear factorization of domains and ranges. Consider Example 3.9 in which it is stated that the property eg:name relates persons, i.e. resources of the type eg:Person to their names which are resources of the type eg:Name.




```
1 eg:name {
2   is [eg:Person → eg:Name]
3 }
```

Example 3.9: The shorthand notation for specifying the type of a property

The shorthand notation in Example 3.9 is expanded as follows.

[2]The actual textual representation of the arrow is -->

```
eg:name {
  rdf:type     → rdf:Property
  rdfs:domain → eg:Person
  rdfs:range   → eg:Name
}
```

When the domain or range is unknown the respective position within the square brackets of the is construct can be left empty. For example, if the domain is unknown eg:name could be specified as follows.

```
eg:name {
  is [ → eg:Name]
}
```

Note that the arrow within the square brackets is obligatory. Otherwise it is not clear which of the domain and range is specified and which is unknown. The shorthand notation could then be expanded by stating that the omitted domain is of type rdf:Resource since every resource is of type rdf:Resource (cf. [15]).

```
eg:name {
  rdf:type     → rdf:Property
  rdfs:domain → rdf:Resource
  rdfs:range   → eg:Name
}
```

Another possibility is to leave the unspecified domain also unspecified in the expansion. When the range is unknown it can be omitted the same way.

```
eg:name {
  is [eg:Person →  ]
}
```

This fragment is equal to the following data term.

```
eg:name {
  rdf:type    → rdf:Property
  rdf:domain → eg:Person
  rdf:range   → rdf:Resource
}
```

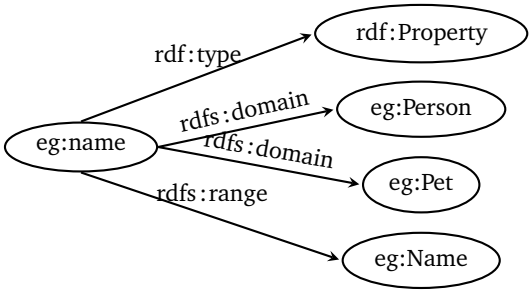
Consistently, both domain and range can be omitted, thus, stating that the subject of the data term is a property which relates resources of type rdf:Resource to resources of type rdf:Resource.

```
eg:name {
  is [ → ]
}
```

This abbreviation can be expanded as follows.

```
eg:name {
  rdf:type     → rdf:Property
  rdfs:domain  → rdf:Resource
  rdfs:range   → rdf:Resource
}
```

As mentioned earlier, domains and ranges can be factorized as introduced in the previous sections. In Example 3.10 factorization is used to state that the property eg:name has the domains eg:Person and eg:Pet.



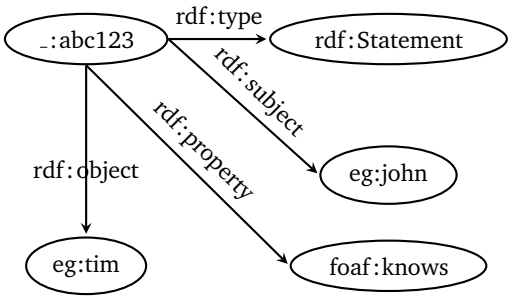

```
eg:name {
 is [
  (eg:Person,eg:Pet) → eg:Name
 ]
}
```

Example 3.10: The shorthand notation for specifying the type of a property

3.3.3 Reification

Most RDF query languages (e.g. SPARQL, Triple and SeRQL) support reification to an extent that they offer syntactic sugar for more convenient writing of reified statements. As mentioned in Section 2.3, this syntactic sugar expresses an implicit blank node. This might be a consequence of giving the reified statement the notion of a "super-node". A super-node represents the reified statement and looks like a special node which may occur at subject as well as object positions. Though the notion of super-nodes is only illustrative, its removal by the corresponding RDF triples leads to a particular shortcoming: The resulting implicit blank node cannot be given a user-defined name, thus, it is not addressable from elsewhere within the RDF graph. Moreover, it cannot be replaced by an URI making program editing unnecessary complicated. In those cases, the shorthand notation for reification is not applicable which makes using the corresponding longhand notations unavoidable. Therefore, the notion of reified statements as super-nodes is not retained for Xcerpt^{RDF}. Instead, the node that represents the reified statement remains explicit. Thus, it can be named either explicitly (i.e. by the user) or implicitly (i.e. by the system) or it can be replaced by an URI. Note that the case of implicit naming of the blank node yields the same result one would get in the query languages mentioned above. Consider the following example in which the statement eg:John{foaf:knows→eg:Tim} is reified with _:abc123 as the reification quad.

According to the example above, the reified statement is written between $< \cdot >$. The shorthand notation above abbreviates the following data term:




```
_:abc123 {
  <eg:john{foaf:knows → eg:tim}>
}
```

Example 3.11: Syntactic sugar for reification in Xcerpt^{RDF}

```
_:abc123  {
  rdf:type       → rdf:Statement
  rdf:subject    → eg:john
  rdf:predicate  → foaf:knows
  rdf:object     → eg:tim
}
```

As the Example 3.11 shows, the shorthand notation for reification does not hide the blank node _:abc123 representing the reification quad, thus, it can be easily replaced by an implicitly named blank node representing the reification quad.

```
_: {
  <eg:john{foaf:knows → eg:tim}>
}
```

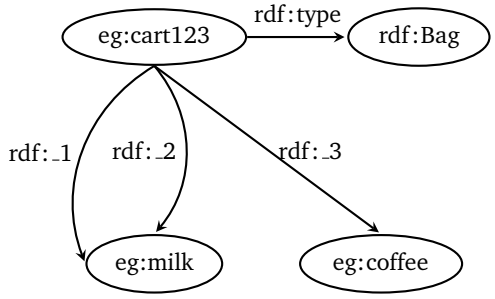
Moreover, it is possible to identify the reification quad by a URI, e.g. eg:reif123 as it is shown below.

```
eg:reif123 {
  <eg:john{foaf:knows → eg:tim}>
}
```

Recapitulatory, when using the shorthand notation in conjunction with explicitly named blank nodes or URIs, the reification quad remains addressable. In these cases, compared to the respective shorthand notations of other query languages, the shorthand notation of Xcerpt^{RDF} is applicable to a greater extent. In conjunction with implicitly named blank nodes, the expressiveness of the shorthand notation is equal to those of current RDF query languages.

3.3.4 Containers and Collections

As mentioned in Section 2.1, RDF supports several collection types. For example, a bag is a collection whose elements are unordered and may comprise duplicates, whereas a sequence can be seen as a vector, i.e. its elements are considered to be ordered. For a motivating example

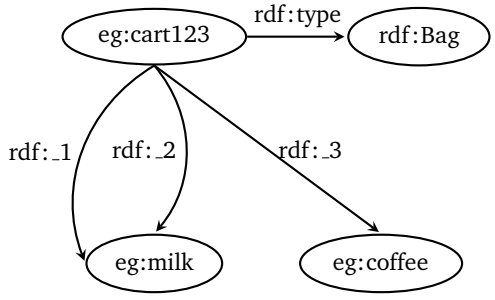



```
1 eg:cart123 {
2   rdf:type → rdf:Bag
3   rdf:_1   → eg:milk
4   rdf:_2   → eg:milk
5   rdf:_3   → eg:coffee
6 }
```

Example 3.12: A shopping cart modelled as a bag (i.e. multiset)

consider a shopping cart to be modelled as a rdf:Bag container.

Shorthand notations for containers are not available in current RDF query languages. This might result from the fact that containers are supposed to be rarely used. However, especially when rdf:Bag containers contain many elements a shorthand notation can be useful. That is why Xcerpt^{RDF} supports shorthand notations for containers. Example 3.12 can then be rewritten as shown in Example 3.13.



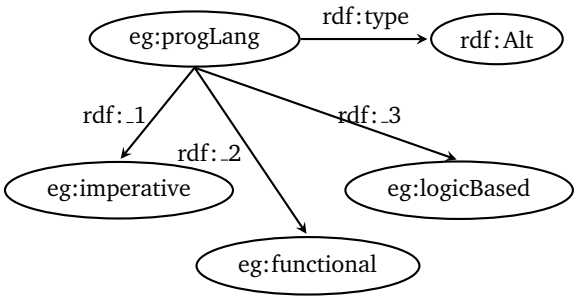

```
1 eg:cart123 {
2   bagOf{eg:milk, eg:milk,
3         eg:coffee}
4 }
```

Example 3.13: A shopping cart modelled as a bag using bagOf{...}

The rdf:Bag container is abbreviated by the shorthand notation bagOf. The elements of the bag are parenthesized in double curly braces and separated by commas. The list of elements within the bagOf-macro may also be composed. In stipulating the use of the appropriate kinds of parentheses, Xcerpt^{RDF} implements the intended semantics of containers as it is described in [15].

For the remaining RDF containers, i.e. rdf:Seq and rdf:Alt containers, respective shorthand notations are also provided by Xcerpt^{RDF}. Consider the following Example 3.14 for rdf:Alt.

Obviously, the Xcerpt^{RDF} data term is equivalent to the following.

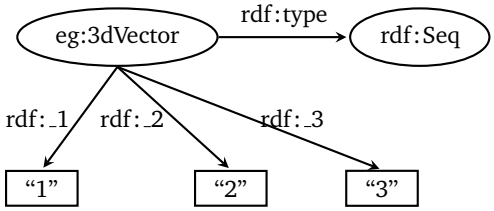



```
eg:progLang {
  altOf{ eg:imperative,
         eg:functional,
         eg:logicBased
       }
}
```

Example 3.14: Types of programming languages modelled as a rdf:Alt container using altOf{...}

```
eg:progLang  {
  rdf:type → rdf:Alt
  rdf:_1   → eg:imperative
  rdf:_2   → eg:functional
  rdf:_3   → eg:logicBased
}
```

As in rdf:Alt-containers the order of elements is irrelevant; the shorthand notation altOf is used in conjunction with curly braces. The rdf:Seq container can be abbreviated by the following shorthand notation.




```
eg:3dVector {
  seqOf["1", "2", "3"]
}
```

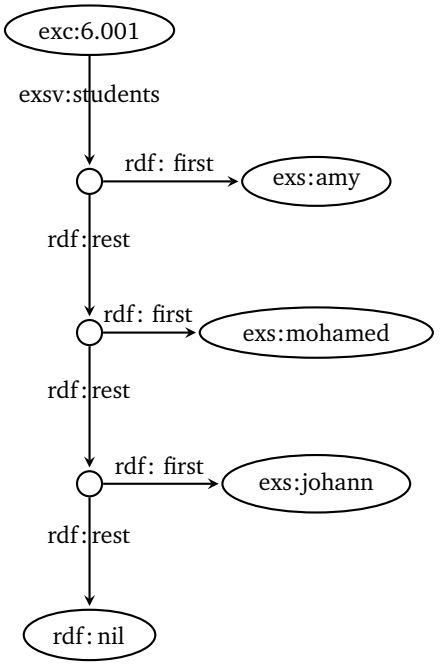
Example 3.15: A three-dimensional vector modelled as a rdf:Seq container using seqOf[...]

The rdf:Seq container is the only container in which the elements' order is considered relevant. To take the order into account, the seqOf-macro is used in conjunction with square brackets. The Xcerpt^{RDF} data term can be equivalently expressed using the RDF-native description.

```
eg:3dVector  {
  rdf:type → rdf:Seq
  rdf:_1   → "1"
  rdf:_2   → "2"
  rdf:_3   → "3"
}
```

Although its structure differs from that of the RDF containers, the collection rdf:List is supported in Xcerpt^{RDF} using listOf the same way. Whereas the elements of RDF containers are

described by rdf:_1, rdf:_2, ..., rdf:_n, lists are built by rdf:first describing the element and rdf:rest recursively describing the rest of the list. A list can be closed by using rdf:nil, i.e. according to the intended semantics of rdf:List only the elements which are specified belong to a list. Consider Example 3.16 which is taken from [19].




```
exc:6.001 {
  exsv:students → _: {
    rdf:first → exs:amy
    rdf:rest  → _: {
      rdf:first → exs:mohamed
      rdf:rest  → _: {
        rdf:first → exs:johann
        rdf:rest  → rdf:nil
      }
    }
  }
}
```

Example 3.16: The example of rdf:List taken from [19]

As can be seen, the notation of rdf:List containers is very cumbersome and confusing. That is why RDF query languages like SPARQL also offer shorthand notations for rdf:List containers. Example 3.17 shows the equivalent Xcerpt^{RDF} data term of the example above using the shorthand notation listOf.

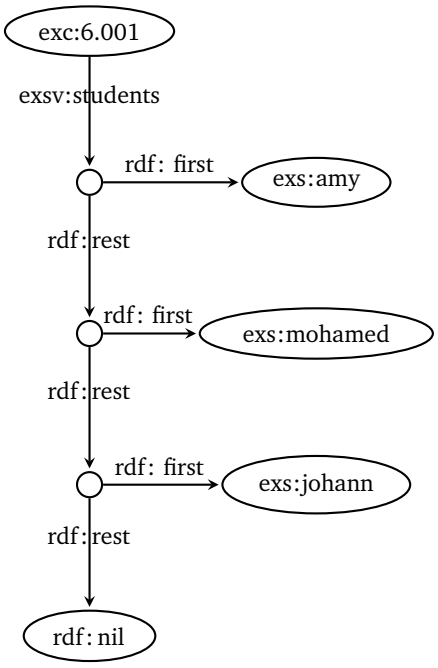
Note that in Xcerpt^{RDF} the notation listOf is used in conjunction with single square brackets having the same meaning as in Xcerpt^{XML}, i.e. it specifies the order of subterms as relevant and the list as being complete.

3.3.5 The EBNF Grammar of Xcerpt^{RDF} Data Terms

Xcerpt^{RDF} data terms can be constructed according to the EBNF-Grammar given in the following listing.

Listing 3.1: EBNF grammar for Xcerpt^{RDF} data terms

```
1 <RDF–DATA>          ::= "RDF–GRAPH" [<URI>] "{" <RDF–STATEMENTS> "}".
2 <RDF–STATEMENTS>    ::= <STATEMENT> { [","] <STATEMENT> }.
3 <STATEMENT>         ::= <SUBJECTS> "{" <SUBTERMS> "}".
```




```
exc:6.001 {
  exsv:students → _: {
    listOf[
      exs:amy, exs:mohamed,
      exs:johann
    ]
  }
}
```

Example 3.17: The example of rdf:List taken from [19] using listOf[...]

```
<SUBTERMS>        ::= <SUBTERM> { [","] <SUBTERM> }.
<SUBTERM>         ::= <CONTAINER>    | <RDF–LIST>     |
                      <TYPE>         | <REIFICATION> |
                      <PROPERTY–OBJ>.
<CONTAINER>       ::= "bagOf"  "{" <TERM> {"," <TERM> } "}" |
                      "seqOf"  "[" <TERM> {"," <TERM> } "]" |
                      "altOf"  "{" <TERM> {"," <TERM> } "}".
<RDF–LIST>        ::= "listOf" "[" <TERM> {"," <TERM> } "]".
<TYPE>            ::= "is" ( <NO–LITERALS> |
                       "["[<NO–LITERALS>] "–->" [<NO–LITERALS>] "]" ).
<REIFICATION>     ::= "<" <SUBJECT> "{" <PROPERTY> "–->" <SUBJECT> "}" ">".
<PROPERTY–OBJ>    ::= <PROPERTIES> "–->" <TERMS>.
<SUBJECTS>        ::= <SUBJECT> |
                      "(" <SUBJECT> "," <SUBJECT> { "," <SUBJECT> } ")".
<SUBJECT>         ::= <URI> | <BNODE>.
<PROPERTIES>      ::= <URI> | "(" <URI> "," <URI> { "," <URI> } ")".
<TERMS>           ::= <TERM> | "(" <TERM> "," <TERM> { "," <TERM> } ")".
<TERM>            ::= <URI> | <BNODE> | <LITERAL> | <STATEMENT>.
<NO–LITERAL>      ::= <URI> | <BNODE> | <STATEMENT>.
<NO–LITERALS>     ::= <NO–LITERAL> | "(" <NO–LITERAL> "," <NO–LITERAL>
                                   { "," <NO–LITERAL> } ")".
<BNODE>           ::= "_:" [ <IDENT> ].
<LITERAL>         ::= <STRING> (   [ <LTAG> ] | [ <LTYPE> ] ) .
<LTYPE>           ::= "^^" <URI>.
<LTAG>            ::= "@" <XML–LANGUAGE–TAG>.
```

29 <URI> ::= "@"<RFC–URI> | <QNAME>.

For abbreviation purposes the productions of several grammar variables are omitted. The production of <XML–LANGUAGE–TAG> is conformable to [20], the production of <RFC–URI> is according to [6] and <QNAME> is according to [25]. <STRING> produces arbitrary quoted strings and <IDENT> produces any legal label.

Note that because of the lines 21 and 22 the grammar is not LL(1).

3.4 Query Terms

Just like in Xcerpt^{XML}, query terms in Xcerpt^{RDF} serve as patterns for selecting data. These patterns may be incomplete and also specify optional and negative subterms with the same notions as given in Section 2.4 and [22]: If a subterm is declared optional, it will not need to be necessarily available; and if a subterm is declared negative, it may not be available within the data selected. Xcerpt^{RDF} also resembles Xcerpt^{XML} in occurrence of label variables. Moreover, regular expressions serve as constraints for the labels within the data selected. Just like in Xcerpt^{XML}, incompleteness is specified by double curly braces. Note that single or double square brackets do not occur as subterm specifications since subterms are always considered unordered in Xcerpt^{RDF}. Label variables are declared the same way as in Xcerpt^{XML} by the keyword var followed by an identifier. Regular expressions may occur at label positions, i.e. as subjects, properties or objects.

Example 3.1 (Query Terms in Xcerpt^{RDF})**.** *In the following example of Xcerpt^{RDF} query terms are given.*

- $var\ X\{\{var\ Y \rightarrow var\ Z\}\}$ *is a pattern for selecting every Xcerpt^{RDF} data term.*
- $var\ X\{\{$ foaf:knows $\rightarrow var\ Z\}\}$ *is a pattern for selecting persons X who know some Y.*
- $var\ X\{\{$ foaf:knows $\rightarrow var\ Z,$ foaf:knows $\rightarrow var\ W\}\}$ *is a pattern for selecting persons X who know at least two different persons Z and W.*
- $var\ X\{\{$ foaf:knows $\rightarrow var\ Z,\ optional$ foaf:mbox $\rightarrow var\ M\}\}$ *is a pattern for selecting persons X and, if available, their email addressed M who know some persons Z.*

✦

Regarding negative subterms, explicit bracketing must be introduced in order to define the scopes of this construct. This is due to the fact that, in contrast to Xcerpt^{XML}, subterms in Xcerpt^{RDF} consist of two parts, the property and the object, thus, it must be specified which of these parts are affected by $without$. This is done by round parentheses which are also used for bracketing subject, property and object lists. Consider the following example in which all possible scopes are listed.

Example 3.2 (Different Positions and Scopes of $without$)**.** *For convenience, namespace prefixes are omitted, X denotes a variable standing for a person.*

- X$\{\{(without$ knows $\rightarrow$ Jim$)\}\}$ *is a pattern for retrieving those persons who do not know Jim, while these persons may be related to Jim by another property.*

- X{{($without$ knows) → Jim}} *is a pattern for retrieving those persons who are related to Jim in a way other than by knowing him. The difference between this pattern and the first pattern is that the persons must be related to Jim.*
- X{{ knows → ($without$ Jim)}} *is a pattern for retrieving persons who know someone other than Jim.* ✦

In the latter case of Example 3.2, the explicit bracketing can be omitted since there is no possibility of confusion.

Regarding the $optional$ construct two scopes are meaningful: Either the entire subterm or the object is within the scope of $optional$. Only the property within the scope of $optional$ makes no sense because whenever a subject and a object are obligatory there must be a property available.

3.4.1 Variables for Concise Bounded Descriptions

Variables may often be bound to blank nodes which, however, only state the existence of a resource. The only further information carried by blank nodes is that two blank nodes with identical names can be considered to be identical. However, the opposite does not hold, i.e. two blank nodes with different names cannot be considered to be different. In order to get more information about blank nodes, their immediate environment, i.e. their concise bounded descriptions [24], can be used to "locate" the blank node in the RDF graph. Moreover, concise bounded descriptions comprise all the direct information about a given resource. For example, the query "Give me all you know about John" can be evaluated by the concise bounded description of the resource representing John. Therefore, in Xcerpt$^{\text{RDF}}$ a new variable type is introduced, standing for the concise bounded description of a resource. A concise bounded description variable – or CBD variable for short – is declared by the keyword $cbdvar$ followed by an identifier and is bound to the CBD of the label to which the variable is bound. Consider the following example.

Example 3.3 (Concise Bounded Description Variables)**.** *Let the following data be given.*
eg:John {{ foaf:knows → _:abc123, foaf:knows → eg:Anna }},
_:abc123{{ foaf:knows → _:xyz321, foaf:knows → eg:Anna }}
The query term $cbdvar\ X${{ foaf:knows → eg:Anna }} *would yield two concise bounded descriptions.*

1. *X is bound to* eg:John *and the concise bounded description of* eg:John *is*
   eg:John {{ foaf:knows → _:abc123, foaf:knows → eg:Anna }},
   _:abc123{{ foaf:knows → _:xyz321, foaf:knows → eg:Anna }}
2. *X is bound to* _:abc123 *and the concise bounded description of* _:abc123 *is*
   _:abc123{{ foaf:knows → _:xyz321, foaf:knows → eg:Anna }} ✦

No RDF query language currently supports concise bounded descriptions (cf. [13]), but in SPARQL (cf. [21]) concise bounded descriptions as a possible mechanism for evaluating the construct DESCRIBE are mentioned explicitly.

Usually, blank nodes are frequently used when prototyping RDF data or when describing RDF containers and collections. However, an exhaustive use of blank nodes results in expensive computations of concise bounded descriptions. In the worst case the concise bounded description of a blank node comprises the complete RDF graph. Thus, as an efficiency consideration it is rational to limit the maximum depth of the computation. As the maximum depth depends on the field of application it is suggested that the user may optionally set this value globally at the beginning of an Xcerpt program. Therefore, the maximum depth of the computation of concise bounded description can be declared by the keyword cbd−depth−counter at the beginning of an Xcerpt program.

For example, the declaration cbd−depth−counter = 23 sets the maximum depth counter to 23 which means that starting from the label for which the concise bounded description is to be computed the maximum path length may not exceed 23 steps.

Within an Xcerpt query it might also be useful to provide means to deviate from the global declaration of the maximum depth counter. This could be done by redefining this value either within the scope of the query (e.g. re-declaring cbd−depth−counter within the WHERE clause of the query) or for each CBD variable declaration. Regarding the latter, it is suggested to add the maximum depth counter in parentheses to the keyword cbdvar. For example, having globally declared cbd−depth−counter = 23 the declaration $cbdvar(10)\ X$ within a query overrides the maximum depth counter for the CBD variable X to 10. For further CBD variables other than X the global depth counter of 23 remains decisive.

3.4.2 Shorthand Notations for Containers in Query Terms

As Xcerpt^{RDF} is based on patterns, shorthand notations, especially for containers, may also occur in query terms. However, shorthand notations for containers within Xcerpt^{RDF} query terms have to be treated differently from data terms because their capabilities described below exceed the expressiveness of RDF.

Example 3.4 (The Problem of Expanding bagOf in Query Terms)**.** *Let the data term* eg:John { bagOf { "a","p","w","l" } } *be given. Furthermore, let the ground query term* eg:John {{ bagOf { "a","l" } }} *be given. According to the intended semantics of rdf:Bag the query term is supposed to match with the data term, since it queries for the elements "a" and "l" which are contained in the data. Now, consider the expansion of both query and data term which yields the following.*

Query Term

```
1  eg:John {{
2    rdf:type → rdf:Bag
3    rdf:_1   → "a"
4    rdf:_2   → "l"
5  }}
```

Data Term

```
1  eg:John {
2    rdf:type → rdf:Bag
3    rdf:_1   → "a"
4    rdf:_2   → "p"
5    rdf:_3   → "w"
6    rdf:_4   → "l"
7  }
```

The expanded query term, however, does not match with the data term, since in the query term the element "l" is connected by the property rdf:_2 whereas in the data term the element "l" is connected by the property rdf:_3. ✦

This problem does not occur when expanding seqOf (due to its ordered nature). Another issue regarding RDF sequences (but also the other RDF containers) is the incompleteness of its elements . There are query scenarios conceivable in which an RDF sequence is queried according to an incomplete specification of its elements and the relative order between them. For example, consider a query of an RDF sequence in which the element "a" shall range before the element "1" independently from the positions of "a" and "1" within the sequence.

Xcerpt^{XML} already offers a way for expressing incompleteness by using double brackets. Thus, double square brackets are adopted to express incomplete specification of RDF container elements. Consider Example 3.5 which contrasts both ways of querying RDF sequences.

Example 3.5. *Let the data term* $a\{$ seqOf$[" x", " y", " z"]\}$ *be given.*

- $var\ X\{\{$ seqOf$[" x", " y", " z"]\}\}$ *matches with the data term.*
- $var\ X\{\{$ seqOf$[" x", " y"]\}\}$ *matches with the data term, since the element list of the sequence within the data term starts with* $" x", " y"$.
- $var\ X\{\{$ seqOf$[" x"]\}\}$ *matches with the data term, since the element list of the sequence within the data term starts with* $" x"$.
- $var\ X\{\{$ seqOf$[" x", " z"]\}\}$ *does not match with the data term.*
- $var\ X\{\{$ seqOf$[" y", " z"]\}\}$ *does not match with the data term.*
- $var\ X\{\{$ seqOf$[[" x", " z"]]\}\}$ *matches with the data term, since in the element list of the sequence within the data term,* $" x"$ *comes before* $" z"$.
- $var\ X\{\{$ seqOf$[[" y", " z"]]\}\}$ *matches with the data term, since in the element list of the sequence within the data term* $" y"$ *comes before* $" z"$. ✦

The different kinds of specifying RDF container elements offered in Xcerpt^{RDF} prohibit the expansion of these shorthand notations within query terms because they exceed the expressiveness of RDF. Moreover, it has been decided to reduce longhand notations of RDF containers within query terms to their respective shorthand notations. This simplifies the simulation of query terms in query terms which is used for deciding subsumption (cf. Section 5.5).

However, making this decision, the so-called "well-formedness" of RDF containers (cf. [19]) becomes an issue. An RDF container is considered to be not well-formed if, among other things, its membership-properties are not numbered subsequently. Consider the following non well-formed RDF container written as Xcerpt^{RDF} data term.

```
1 eg:shoppingCart  {
2   rdf:type    → rdf:Bag
3   rdf:_1      → eg:Milk
4   rdf:_5      → eg:Coffee
5   rdf:_100    → eg:Milk
6 }
```

The rdf:Bag container lacks the member properties rdf:_2 to rdf:_4 and rdf:_6 to rdf:_99. In data terms non well-formedness of RDF containers plays a minor role so that Xcerpt^{RDF} only offers shorthand notations for well-formed containers. However, in query terms non well-formed RDF containers are quite eligible. For example the query term

```
1 var X {{
2   rdf:type   → rdf:Seq
3   rdf:_1     → eg:anna
4   rdf:_52    → eg:john
5 }}
```

queries for any RDF sequence in which eg:anna is at the first position and eg:john is at the 52nd position. No restrictions are given for the remaining elements.

However, the missing specifications of the remaining container elements become problematic when the container subterms shall be abbreviated by the corresponding shorthand notation in which well-formedness is required. A solution would be to pad the element list of the shorthand notation by blank nodes or wildcards. Additionally, a warning could be issued to give the user the opportunity to review his query. Nonetheless, such queries are conceivable and not necessarily flawed. In these cases, this approach would slow down the evaluation considerably.

Another more robust approach is to adopt the notion of positional variables as introduced in Xcerpt^{XML}. Elements of RDF containers within query terms can then be extended by a specification of their positions (introduced by the keyword pos). In the remaining thesis container elements with positions specifications will be referred to as extended container elements. For example, the query term given above could be reduced to the following query term.

```
1 var X {{
2   seqOf[[pos 1 eg:anna, pos 52 eg:john]]
3 }}
```

The double brackets of seqOf specify incompleteness of the elements. Thus, the element specification by brackets in conjunction with the position specification of container elements also permits the querying of non well-formed containers. For example, var X {{ seqOf[pos 1 a, pos 4 w] }} matches with every sequence to which the subterms rdf:_1 → a and rdf:_4 → w exclusively belong.

Mixing of elements and extending elements within an Xcerpt^{RDF} container is conceivable and certainly useful. By mixing, it would be possible to query extended elements on the basis of the actual data base whereas for the unextended elements the intended semantics is used. However, the evaluation of Xcerpt^{RDF} containers would become very complex so that (for now) it is suggested to disallow mixing both kinds of elements within an Xcerpt^{RDF} container.

However, the interpretation of the brackets for container elements as described above, conflicts with their meanings in Xcerpt^{XML}. The meanings of the brackets originate from the evaluation of the longhand notations of containers in conjunction with the inherent incompleteness of query terms. Consider Example 3.5: The single square brackets of RDF containers within query

terms make the RDF container match with the data if their element lists are prefixes of the element lists of the RDF containers within the data terms. Reading single square brackets from the Xcerpt^{XML} perspective means that the element list is, in particular, considered as complete. Thus, in Example 3.5 only the first query would succeed.

As a remedy, it is suggested to retain the meanings of the various kinds of brackets as given in Xcerpt^{XML} in order to stay conform with Xcerpt^{XML}. Furthermore, it is suggested to elaborate on other ways of bracketing the elements in order to express the meanings given in Example 3.5. Unfortunately, due to time limitations, this topic cannot be covered in this thesis.

Summarizing the above, the different kinds of brackets and elements allow querying RDF containers not only on the basis of the intended semantics but also on the basis of the actual specification of the data. No current RDF query language supports the querying of RDF containers to that extent. The evaluation of unexpanded containers in Xcerpt^{RDF} conforming to Xcerpt^{XML} is discussed in Chapter 5.

3.4.3 The EBNF Grammar of Xcerpt^{RDF} Query Terms

The EBNF grammar of Xcerpt^{RDF} query terms builds upon the grammar of Xcerpt^{RDF} data terms. The main differences are the introduction of label and CBD variables, positive and negative subterms and labels and additional bracketing for RDF containers and collections.

Listing 3.2: The EBNF Grammar for Query Terms

```
1  <QUERYTERM>          ::=
2      <QANYSUBJECTS> "{{" <QANYSUBTERMS> "}}".
3
4  <QANYSUBTERMS>       ::=
5      <QANYSUBTERM> ("," | "\n" ) { <QANYSUBTERM> }.
6
7  <QANYSUBTERM>        ::=
8      <QNEGSUBTERM> | <QOPTSUBTERM> | <QSUBTERM>.
9
10 <QNEGSUBTERM>        ::=
11     "(" "without" <QSUBTERM> ")".
12
13 <QOPTSUBTERM>        ::=
14     "(" "optional" <QSUBTERM> ")".
15
16 <QSUBTERM>           ::=
17     <QCONTAINER> | <QRDF-LIST>       |
18     <QTYPE>      | <QREIFICATION>    |
19     <QPROPERTY-OBJ> .
20
21 <QCONTAINER>         ::=
22     "bagOf"  ( <UNORDINCPL-LIST> | <UNORDCPL-LIST> ) |
23     "altOf"  ( <UNORDINCPL-LIST> | <UNORDCPL-LIST> ) |
24     "seqOf"  ( <ORDINCPL-LIST>   | <ORDCPL-LIST> ).
25
26 <QRDF-LIST>          ::=
```

```
27      "listOf" ( <ORDINCPL–LIST>    | <ORDCPL–LIST> ).
28
29  <QTYPE>              ::=
30      "is" ( <QANYNO–LITERALS> |
31           "[" [<QANYNO–LITERALS>] "–->" [<QANYNO–LITERALS>] "]" ).
32
33  <QREIFICATION>       ::=
34      "<" <QANYSUBJECT> "{"
35          <QANYPROPERTY> "–->" <QANYSUBJECT> "}" ">".
36
37  <UNORDINCPL–LIST>    ::=
38      "{{" <QELEMENTLIST> "}}".
39
40  <UNORDFCPL–LIST>     ::=
41      "{"  <QELEMENTLIST> "}".
42
43  <ORDINCPL–LIST>      ::=
44      "[[" <QELEMENTLIST> "]]".
45
46  <ORDCPL–LIST>        ::=
47      "[" <QELEMENTLIST> "]".
48
49  <QELEMENTLIST>       ::=
50      <QANYTERM> {"," <QANYTERM>}.
51
52  <QPROPERTY–OBJ>      ::=
53      <QANYPROPERTIES> "–->" <QANYTERMS>.
54
55
56  <QANYSUBJECTS>       ::=
57      <QANYSUBJECT> |
58      "(" <QANYSUBJECT> "," <QANYSUBJECT> { "," <QANYSUBJECT> } ")".
59
60  <QSUBJECTS>          ::=
61      <QSUBJECT>       |
62      "(" <QSUBJECT> "," <QSUBJECT> { "," <QSUBJECT> } ")".
63
64  <QANYSUBJECT>        ::=
65      <QNEGSUBJECT> | <QOPTSUBJECT> | <QSUBJECT>.
66
67  <QNEGSUBJECT>        ::=
68      "(" "without" <QSUBJECTS> ")".
69
70  <QOPTSUBJECT>        ::=
71      "(" "optional" <QSUBJECTS> ")".
72
73  <QSUBJECT>           ::=
74      <VAR>            | <CBD–VAR> | <SUBJECT>.
75
76  <QANYPROPERTIES>     ::=
77      <QANYPROPERTY> |
```

```
78      "(" <QANYPROPERTY> "," <QANYPROPERTY> { "," <QANYPROPERTY> } ")".
79
80  <QPROPERTIES>       ::=
81      <QPROPERTY> | "(" <QPROPERTY> { "," <QPROPERTY> } ")".
82
83  <QANYPROPERTY>      ::=
84      <QPROPERTY> | <QNEGPROPERTY> | <QOPTPROPERTY>.
85
86  <QPROPERTY>         ::=
87      <VAR> | <URI>.
88
89  <QNEGPROPERTY>      ::=
90      "(" "without" <QPROPERTY> ")".
91
92  <QOPTPROPERTY>      ::=
93      "(" "optional" <QPROPERTY> ")".
94
95  <QANYTERMS>         ::=
96      <QANYTERM> |
97      "(" <QANYTERM> "," <QANYTERM> { "," <QANYTERM> } ")".
98
99  <QTERMS>            ::=
100     <QTERM>     |
101     "(" <QTERM> "," <QTERM> { "," <QTERM> } ")".
102
103 <QANYTERM>          ::=
104     <QTERM>     | <QNEGTERM> | <QOPTTERM>.
105
106 <QTERM>             ::=
107     <VAR>       | <CBD-VAR>   | <TERM>.
108
109 <QNEGTERM>          ::=
110     "(" "without" <QTERM> ")".
111
112 <QOPTTERM>          ::=
113     "(" "optional" <QTERM> ")".
114
115 <QANYNO-LITERALS> ::=
116     <QNO-LITERAL> |
117     "(" <QANYNO-LITERAL> "," <QANYNO-LITERAL>
118      { "," <QANYNO-LITERAL> } ")"
119
120 <QNO-LITERALS>      ::=
121     <QNO-LITERAL> | "(" <QNO-LITERAL> "," <QNO-LITERAL>
122                       { "," <QNO-LITERAL> } ")"
123                   | <QNO-LITERALS>.
124
125 <QANYNO-LITERAL>  ::=
126     <QNEGNO-LITERAL> | <QOPTNO-LITERAL> | <QNO-LITERAL>.
127
128 <QNEGNO-LITERAL>  ::=
```

```
129     "(" "without"  <QNO–LITERAL> ")".
130
131 <QOPTNO–LITERAL>  ::=
132     "(" "optional" <QNO–LITERAL> ")".
133
134 <QNO–LITERAL>     ::=
135     <VAR>| <CBD–VAR> | <NO–LITERAL>.
136
137 <VAR>             ::= "var"    <IDENT>.
138
139 <CBD–VAR>         ::= "cbdvar" <IDENT>.
```

3.4.4 Xcerpt^{RDF} Queries - Connectives for Query Terms

Just like in Xcerpt^{XML}, query terms can be connected to queries by using the binary connectives and{...}, or{...} and the unary connective not{...} in a straightforward fashion. For the sake of completeness the EBNF grammar for Xcerpt^{RDF} queries is given as follows.

Listing 3.3: The EBNF Grammar for Xcerpt^{RDF} Queries

```
1
2 <QUERY>       ::= "RDF–GRAPH" ( [<URI>] "{" <QUERY–BODY> "}" |
3                     "{" [<INPUT–RESOURCE>] <QUERY–BODY> "}" ).
4 <QUERY–BODY> ::= <QUERYTERM> |
5                  "and" "{" <QUERY–BODY> "," <QUERY–BODY> "}" |
6                  "or"  "{" <QUERY–BODY> "," <QUERY–BODY> "}" |
7                  "not" <QUERY–BODY>.
```

The Section 3.7 extensively illustrates the writing of queries using these connectives.

3.5 Construct Terms – The Grouping Construct all

Construct terms are used to create new data, thereby consuming the bindings which result from the evaluation of queries. Therefore, it is necessary that variables which occur in a construct term of a rule also occur in the query part of the rule. This is usually referred to as range restrictedness (cf. [22]).

The grammar of construct terms basically complies with the grammar of data terms (cf. Section 3.3.5). However, also variables may occur as labels within construct terms. Furthermore, the grouping constructs all and some can be used to group results according to free variables as described in [22] (also cf. Section 2.4). It is also suggested that the explicit grouping of results using the construct group by {<Variables>} is also available in Xcerpt^{RDF}. Note that explicit grouping allows a grouping of the results according to variables which are not within the scope of the grouping constructs all and some.

In contrast to Xcerpt^{XML}, similar to the constructs optional and without, grouping constructs have several scopes which must be specified by explicit bracketing.

Example 3.6 (Scopes of the Grouping Construct all)**.** *The scopes and the results of the grouping construct all are given for the term* $var\ S\{var\ P\ \rightarrow\ var\ O\}$.

1. $(all\ var\ S\ \{var\ P\ \rightarrow\ var\ O\})$ *groups all possible instances of* $var\ S\ \{var\ P\ \rightarrow\ var\ O\}$ *which result from the different bindings of the variables* S*,* P *and* O*. Such usage leads to triples as answers.*

2. $(all\ var\ S)\{var\ P\ \rightarrow\ var\ O\}$ *groups all possible instances of* $var\ S$ *for each binding of* P *and* O*. This results in subject lists.*

3. $var\ S\ \{(all\ var\ P\ \rightarrow\ var\ O)\}$ *groups all possible instances of* $var\ P\ \rightarrow\ var\ O$ *for each binding of the variable* S *which leads to property-object lists.*

4. $var\ S\ \{(all\ var\ P)\ \rightarrow\ var\ O\}$ *groups all possible instances of* $var\ P$ *for each binding of the variables* S *and* O *which yields property lists.*

5. $var\ S\ \{var\ P\ \rightarrow\ (all\ var\ O)\}$ *groups all possible instances of* $var\ O$ *for each binding of the variables* S *and* P *which yields object lists.* ✦

As in Xcerpt^{RDF} occurrences of labels are not distinguished, in contrast to Xcerpt^{XML}, only few scenarios exist in which the use of the all grouping construct yields results different from construct terms without this grouping construct. To see this, consider the following Example 3.7.

Example 3.7 (Equivalent Construction of Data in Xcerpt^{RDF} Independent from the Grouping Construct all)**.** *Consider the following variable binding given as a set of substitutions* $\{\ \{var\ S \mapsto$ eg:john, $var\ P \mapsto$ eg:knows, $var\ O \mapsto$ eg:jim$\}$, $\{var\ S \mapsto$ eg:john, $var\ P \mapsto$ eg:name, $var\ O \mapsto$ "John"$\}$, $\{var\ S \mapsto$ eg:anna, $var\ P \mapsto$ eg:knows, $var\ O \mapsto$ eg:john$\}$, $\{var\ S \mapsto$ eg:anna, $var\ P \mapsto$ eg:name, $var\ O \mapsto$ "Anna"$\}$ $\}$*. We compare the data created by the following construct terms.*

var S$\{$**var P** $\rightarrow$ **var O**$\}$	**var S**$\{($**all var P** $\rightarrow$ **var O**$)\}$
eg:john{eg:knows → eg:jim} eg:john{eg:name → "John"}	eg:john{eg:knows → eg:jim, eg:name → "John"}
eg:anna{eg:knows → eg:john} eg:anna{eg:name → "Anna"}	eg:anna{eg:knows → eg:john, eg:name → "Anna"}

Comparing both columns shows that the use of the all *grouping construct does not lead to different results, except the fact that the data terms are factorized.* ✦

Although the construct terms in Example 3.7 yield syntactical different data terms, the results describe identical RDF graphs.

However, one case in which the grouping construct all makes sense or is even necessary is the construction of well-formed RDF containers. Therefore, consider the following Example 3.8.

Example 3.8 (The Grouping Construct all in Conjunction with the Construction of RDF Containers)**.** *Let the substitution set* $\{\ \{var\ X \mapsto a\}, \{var\ X \mapsto b\}, \{var\ X \mapsto c\}\ \}$ *be given. Consider the following construct terms.*

eg:b{**bagOf**{**var X**}}	**eg:b**{**bagOf**{(**all var X**)}}
eg:b{$bagOf\{a\}$} eg:b{$bagOf\{b\}$} eg:b{$bagOf\{c\}$}	eg:b{$bagOf\{a, b, c\}$}

The resulting data terms in the left column is equivalent to the data term

```
eg:b {
  rdf:type → rdf:Bag
  rdf:_1   → a
  rdf:_1   → b
  rdf:_1   → c
}
```

which describes a non well-formed RDF container. The resulting data term in the right column, however, is equivalent to the data term

```
eg:b {
  rdf:type → rdf:Bag
  rdf:_1   → a
  rdf:_2   → b
  rdf:_3   → c
}
```

which describes a well-formed RDF container. ✦

Furthermore, grouping with all makes sense in conjunction with anonymous blank nodes. Consider Example 3.9.

Example 3.9 (The Grouping Construct all in Conjunction with Anonymous Blank Nodes)**.** *Let the RDF graph with the URL http://www.example.org/cities.rdf be as follows.*

```
RDF–GRAPH {
 (eg:munich,
  eg:cologne,
  eg:berlin) {eg:cityOf → eg:germany}

 (eg:paris,
  eg:lyon)   {eg:cityOf → eg:france }
}
```

The following Xcerpt program queries all cities and construct data terms in which the cities are grouped according to the countries to which they belong.

```
CONSTRUCT RDF–GRAPH {
 _: {(all eg:inSameCountry → var City)} group by {var Country}
}
FROM RDF–GRAPH @http://www.example.org/cities.rdf {
 var City {{ eg:cityOf → var Country }}
}
END
```

The construct term constructs a new blank node for each countries. If we assume, that these constructed blank nodes were _:anonymous122 *and* _:anonymous349 *the Xcerpt program above yields the following data terms.*

```
RDF–GRAPH {
  _:anonymous122 {
    eg:inSameCountry → eg:munich
    eg:inSameCountry → eg:cologne
    eg:inSameCountry → eg:berlin
  }

  _:anonymous349 {
    eg:inSameCountry → eg:paris
    eg:inSameCountry → eg:lyon
  }
}
```

Without the all *construct only one blank node would have been constructed. Thus, using this grouping construct in combination with anonymous blank nodes leads to different answers than one gets when omitting it.* ✦

The discussion shows that the grouping construct all makes only sense when it is used in conjunction with RDF containers or anonymous blank nodes. In other cases the results are syntactically different but represent the same RDF graph.

3.6 From Xcerpt^{XML} to Xcerpt^{RDF} Back and Forth

Xcerpt^{XML} and Xcerpt^{RDF} support two different data models, the rooted node labelled tree structure of semi-structured data and the unrooted node labelled and edge labelled directed graph structure of RDF respectively. Moreover, other differences exist between XML and RDF. For example, Xcerpt^{RDF} provides other language constructs than Xcerpt^{XML}. These differences necessitate the separation of both worlds and, hence, Xcerpt^{XML} and Xcerpt^{RDF}. This is achieved by the language construct RDF−GRAPH which embraces Xcerpt^{RDF} terms as follows.

```
RDF–GRAPH {
  eg:john {
    foaf:knows → _:xyz345 {
      foaf:name → "Jim"
    }
  }
}
```

In Xcerpt^{RDF} rules, query terms are usually associated with input resources. For example, all triples of an RDF graph with the URL http://www.example.org/myfriends.rdf are queried as illustrated in the following listing.

```
CONSTRUCT
  ...
FROM
RDF-GRAPH {
  in {
    resource[http://www.example.org/myfriends.rdf],
    var X {{ var Y → var Z }}
  }
}
END
```

A shorter way to specify input resources, thus, abbreviating the specification as described above is to declare the graph URL following the keyword RDF−GRAPH. This leads to the shorthand notation as given below.

```
CONSTRUCT
  ...
FROM RDF-GRAPH @http://www.example.org/myfriends.rdf {
  var X {{ var Y → var Z }}
}
END
```

This notation also expresses in a more intuitive way that the RDF graph which is retrievable at the location http://www.example.org/myfriends.rdf shall be queried.

The specification of output resources appearing in construct terms of Xcerpt^{RDF} goals could be written in the same way. Consider the following construction part of an Xcerpt^{RDF} goal.

```

GOAL RDF-GRAPH {
  out {
    resource[file:///home/alex/myfriends.rdf],
    all _: { var Y → var Z }
  }
}
FROM
  ...
END
```

Regarding output resources, it is suggested to use the same shorthand notation as introduced for the specification of input resources.

```
G0AL RDF-GRAPH @file:///home/alex/myfriends.rdf {
  all _: { var Y → var Z }
}
```

Note that although the specifications are written in the same way, their meanings differ. In query parts the resource URI specifies the input resource, whereas in construct parts the resource URI specifies the output resource.

3.7 Xcerpt by Example

This section shows how Xcerpt^{RDF} can be used in conjunction with Xcerpt^{XML} to query RDF data and transfrom RDF to XML based data such as HTML. Therefore, consider the example RDF graph as illustrated in Figure 3.1.

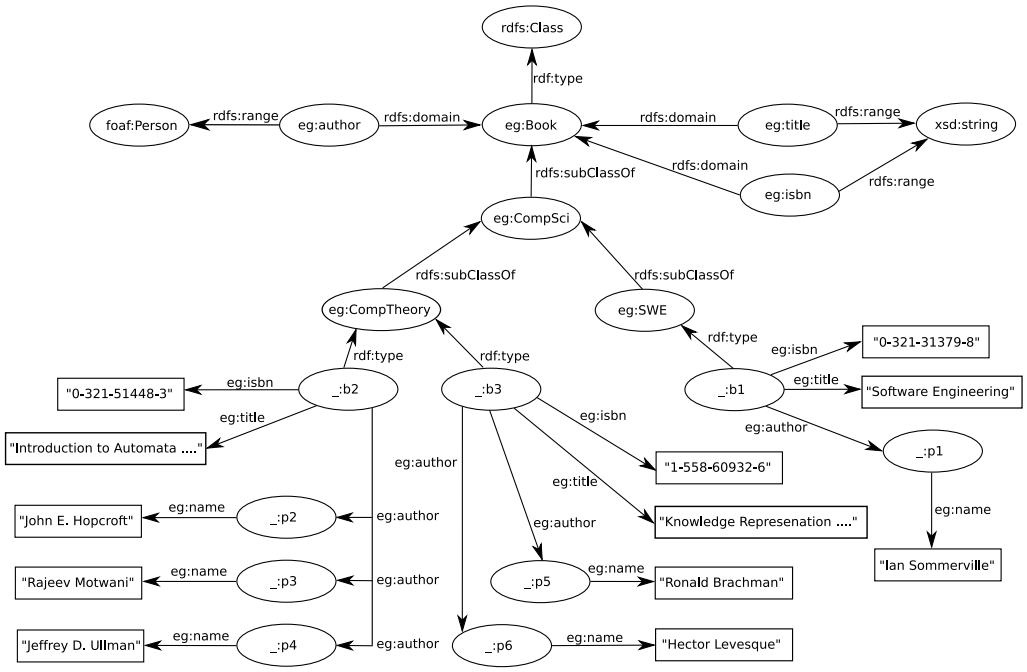


Figure 3.1: The RDF graph example used in this section. The book titles are abbreviated. Also, for simplicity, the figure omits the descriptions of eg:CompTheory, eg:SWE and eg:Math being of type rdfs:Class, and eg:author, eg:isbn and eg: title being of type rdfs:Property.

Using the shorthand notations introduced in the previous sections the graph in Figure 3.1 can be written in Xcerpt^{RDF} as the following listing shows.

Listing 3.4: The representation of the RDF graph

```
1
2 RDF-GRAPH
3 {
4   (eg:Book,
5    eg:CompSci,
6    eg:CompTheory,
7    eg:SWE ) {is rdfs:Class}
8
9   (eg:CompTheory, eg:SWE) { rdfs:subClassOf → eg:CompSci }
```

```
10
11    eg:author { is [foaf:Person → eg:Book] }
12    eg:title  { is [eg:Book     → xsd:string] }
13    eg:isbn   { is [eg:Book     → xsd:string] }
14
15    _:b1 {
16      is eg:SWE
17      eg:author    → _:p1 { eg:name → "Ian Sommerville" }
18      eg:title     → "Software Engineering"
19      eg:isbn      → "0-321-31379-8"
20    }
21
22    _:b2 {
23     is eg:CompTheory
24     eg:author     → _:p2 { eg:name → "John E. Hopcroft"  }
25     eg:author     → _:p3 { eg:name → "Rajeev Motwani"    }
26     eg:author     → _:p4 { eg:name → "Jeffrey D. Ullman" }
27     eg:title      → "Introduction to Automata Theory,
28                       Languages and Computation"
29     eg:isbn       → "0-321-51448-3"
30    }
31
32    _:b3 {
33     is eg:CompTheory
34     eg:author     → _:p5 { eg:name → "Ronald Brachman" }
35     eg:author     → _:p6 { eg:name → "Hector Levesque" }
36     eg:title      → "Knowledge Representation and Reasoning"
37     eg:isbn       → "1-558-60932-6"
38    }
39  }
```

Furthermore, consider that the RDF graph is located at www.example.org/mybooks.rdf.

The first Xcerpt program illustrates the querying of RDF. It constructs a new RDF graph in which the authors of a book are listed within an RDF bag container. The resulting RDF graph is saved as file :///home/alex/authors_in_bags.rdf.

Listing 3.5: Transforming RDF

```
1
2   GOAL RDF-GRAPH @file:///home/alex/authors_in_bags.rdf {
3     var Book {
4       eg:authors → bagOf{ all var Author }
5       var X      → var Y
6     }
7   }
8   FROM RDF-GRAPH @http://www.example.org/mybooks.rdf {
9     var Book {{
10      eg:author → var Author
11      var X     → var Y
12    }}
13  }
```

```
14 END
```

The resulting RDF graph is the following.

Listing 3.6: The resulting RDF graph of the transformation as given above

```
1
2  RDF-GRAPH
3  {
4    _:b1 {
5      eg:authors → bagOf{_:p1}
6      rdf:type   → eg:SWE
7      eg:title   → "Software Engineering"
8      eg:isbn    → "0-321-31379-8"
9    }
10
11   _:b2 {
12    eg:authors → bagOf{_:p2, _:p3, _:p4}
13    rdf:type   → eg:CompTheory
14    eg:title   → "Introduction to Automata Theory,
15                  Languages and Computation"
16    eg:isbn    → "0-321-51448-3"
17   }
18
19   _:b3 {
20    eg:authors → bagOf{_:p5, _:p6}
21    rdf:type   → eg:CompTheory
22    eg:title   → "Knowledge Representation and Reasoning"
23    eg:isbn    → "1-558-60932-6"
24   }
25 }
```

The following Xcerpt program determines the titles, authors and ISBN numbers of each book and returns a HTML file in which these pieces of information are listed within a table. This example program shows the ease of transforming RDF data into semi-structured data in Xcerpt.

Listing 3.7: Transforming RDF to HTML

```
1
2  GOAL
3   out {
4     resource{"file:///home/alex/mybooks-as-html.html"},
5     html [
6       body[
7         table[
8           tr [ th["Title"], th["Author(s)"], th["ISBN Number"] ],
9           tr [
10            td[ var Title ],
11            td[ all p[var Author] ],
12            td[ var Isbn ]
13          ]
14        ]
15      ]
```

```
16      ]
17    }
18  FROM RDF-GRAPH @http://www.example.org/mybooks.rdf {
19    var Book {{
20      eg:title  → var Title
21      eg:author → var Author
22      eg:isbn   → var Isbn
23    }}
24  }
25  END
```

This query yields the answer as given below.

Listing 3.8: The resulting HTML file

```
1
2   <html>
3    <body>
4      <table>
5        <tr><th>Title</th><th>Author(s)</th><th>ISBN</th></tr>
6        <tr>
7          <td>Software Engineering</td>
8          <td>Ian Sommerville</td>
9          <td>0-321-31379-8</td>
10       </tr>
11       <tr>
12         <td>Introduction to Automata Theory,
13             Languages and Computation</td>
14         <td><p>John E. Hopcroft</p>
15             <p>Rajeev Motwani</p>
16             <p>Jeffrey D. Ullman</p>
17         </td>
18         <td>0-321-51448-3</td>
19       </tr>
20       <tr>
21         <td>Knowledge Representation and Reasoning</td>
22         <td><p>Ronald Brachman</p>
23             <p>Hector Levesque</p>
24         </td>
25         <td>1-558-60932-6</td>
26       </tr>
27     </table>
28   </body>
29  </html>
```

Another way of crossing the border between RDF meta-data and semi-structured data is the possibility to combine the querying of RDF data with the querying of XML based data. Therefore, consider an HTML page with the URL http://www.example.org/price-list.html which contains the books including their prizes. Additionally, the table cells which contain the ISBN numbers are tagged by the attribute class="isbn", and the cells containing the prizes are tagged by the attribute class="prize".

Listing 3.9: The sample prize list at the URL http://www.example.org/price−list.html

```
1
2  <html>
3   <body>
4     <table>
5       <tr>
6        <th>Title</th><th>Author(s)</th><th>ISBN</th><th>Price/EUR</th>
7       </tr>
8       <tr>
9         <td>Software Engineering</td>
10        <td>Ian Sommerville</td>
11        <td class="isbn">0−321−31379−8</td>
12        <td class="prize">67,95</td>
13      </tr>
14      <tr>
15        <td>Introduction to Automata Theory,
16            Languages and Computation</td>
17        <td>John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman</td>
18        <td class="isbn">0−321−51448−3</td>
19        <td class="prize">89,99</td>
20      </tr>
21      <tr>
22        <td>Knowledge Representation and Reasoning</td>
23        <td>Ronald Brachman, Hector Levesque</td>
24        <td class="isbn">1−558−60932−6</td>
25        <td class="prize">74,85</td>
26      </tr>
27    </table>
28  </body>
29 </html>
```

The following Xcerpt program shows how to transfer the prizes given in the HTML table to the RDF graph. The program selects the respective books according to their ISBN numbers.

Listing 3.10: Adding the prize information of the prize-list to the RDF graph

```
1
2  CONSTRUCT RDF−GRAPH {
3   var Book {
4     eg:isbn  → var Isbn
5     eg:prize → var Prize
6     var P    → var O
7   }
8  }
9  FROM
10  and {
11    RDF−GRAPH @http://www.example.org/mybooks.rdf {
12      var Book {{
13        eg:isbn → var Isbn
14        (optional var P → var O)
15      }}
16    },
```

```
17      in {
18        resource{http://www.example.org/price-list.html},
19        desc tr {{
20          td {{
21            attributes{ class{"isbn"} },
22            var Isbn
23          }},
24          td {{
25            attributes{ class{"prize"} },
26            var Prize
27          }}
28        }}
29      }
30    }
31  END
```

Note that according to line 14 further property-object pairs are declared optional. Thus, only the ISBN number of a book is required. The program would answer this query as stated below.

Listing 3.11: The resulting RDF graph

```
1  RDF-GRAPH
2  {
3    _:b1 {
4      eg:isbn   → "0-321-31379-8"
5      eg:prize  → "67,95"
6      rdf:type  → eg:CompTheory
7      eg:author → _:p1
8      eg:title  → "Software Engineering"
9    }
10
11   _:b2 {
12     eg:isbn   → "0-321-51448-3"
13     eg:prize  → "89,99"
14     rdf:type  → eg:CompTheory
15     eg:author → _:p2
16     eg:author → _:p3
17     eg:author → _:p4
18     eg:title  → "Introduction to Automata Theory,
19                  Languages and Computation"
20   }
21
22   _:b3 {
23     eg:isbn   → "1-558-60932-6"
24     eg:prize  → "74,85"
25     rdf:type  → eg:CompTheory
26     eg:author → _:p5
27     eg:author → _:p6
28     eg:title  → "Knowledge Representation and Reasoning"
29   }
30 }
```

Furthermore, Xcerpt programs for example can be used in order to compute the transitive closure of the rdfs:subClassOf-relationship and, thus, can be used to implement the RDF Semantics (cf. [15]). The following suggested Xcerpt program could be used for this purpose.

Listing 3.12: Computing the transitive closure of the rdfs:subClassOf-relationship

```
1
2  CONSTRUCT RDF–GRAPH {
3    var Concept1 {
4      rdfs:subClassOf → var Concept3
5    }
6  }
7  FROM RDF–GRAPH @http://www.example.org/mybooks.rdf {
8    and {
9      var Concept1 {{
10       rdfs:subClassOf → var Concept2
11     }},
12     var Concept2 {{
13       rdfs:subClassOf → var Concept3
14     }}
15   }
16 }
17 END
```

This program yields the following data terms if it is evaluated with forward chaining (otherwise it does not terminate).

Listing 3.13: The answer of the program above

```
1
2  RDF–GRAPH
3  {
4    eg:CompTheory { rdfs:subClassOf → eg:Book }
5    eg:SWE { rdfs:subClassOf → eg:Book }
6  }
```

3.8 Digression: Specifying Injectivity in Xcerpt^{XML} by Delimiters

Discussing about the syntax of Xcerpt^{RDF} it has been pointed out that delimiters like the semicolon can also be used in Xcerpt^{XML} in order to specify whether subterms are matched injectively or not. In Xcerpt^{XML} the use of the triple brackets {{{, }}} or [[[,]]] specifies that the subterms are to be mapped revoking injectivity. This is illustrated in Example 3.10.

Example 3.10 (Revoking the Injectivity of Subterm-Mapping)**.** *The Xcerpt^{XML} term $a\{\{\ b\{\{\ \}\},\ b\{\{\ \}\}\ \}\}$ cannot be mapped to the term $a\{\{\ b\{\{\ \}\}\ \}\}$ since injectivity is required. The term $a\{\{\{\ b\{\{\ \}\},\ b\{\{\ \}\}\ \}\}\}$, however, can be mapped to $a\{\{\ b\{\{\ \}\}\ \}\}$, since the triple brackets specify, that both occurrences of $b\{\{\ \}\}$ may be mapped to the same subterm.* ✦

Instead of using the triple brackets, semicolons can be used in order to revoke injectivity. The subterm order and completeness is further on specified by the respective single or double brackets. Thus, with semicolons the term $a\{\{\{\ b\{\{\ \}\},\ b\{\{\ \}\}\ \}\}\}$ could be written as $a\{\{\ b\{\{\ \}\};\ b\{\{\ \}\}\ \}\}$.

Note the semicolon between the two $b\{\{\ \}\}$. The advantage of using semicolons is that for some subterms the injectivity can be revoked, whereas for the remaining subterms, injectivity is still required. This enables to specify injectivity more fine-grained than be done by triple brackets.

Example 3.11 (Revoking Injectivity of Subterm-Mapping with Semicolons)**.** *When assuming that a semicolon binds stronger than a comma, then the term* $a\{\{\ b\{\{\ \}\};\ b\{\{\ \}\},\ c\{\{\ \}\},\ c\{\{\ \}\}\ \}\}$ *specifies that the both subterms of the form* $b\{\{\ \}\}$ *may be mapped to the same subterm, whereas the subterms of the form* $c\{\{\ \}\}$ *have to be mapped to two distinct subterms. Thus, the term could be mapped to* $a\{\{\ b\{\{\ \}\},\ c\{\{\ \}\},\ c\{\{\ \}\}\ \}\}$*, but not to* $a\{\{\ b\{\{\ \}\},\ c\{\{\ \}\}\ \}\}$*.* ✦

Chapter 4

Normalizing Xcerpt^{RDF} Terms

Normalizing Xcerpt^{RDF} terms is necessary to keep their evaluation clear and simple. The various possibilities to express knowledge in Xcerpt^{RDF} makes normalization necessary.

Xcerpt^{RDF} data terms correspond to pure RDF data although the various shorthand notations for RDF constructs discussed in Chapter 3 exceed the expressivity of RDF. However, in this thesis it has been decided to retain RDF expressivity as far as possible. As a consequence, Xcerpt^{RDF} shorthand notations are expanded within data terms, thus, in normalized data terms only property-object pseudo-terms occur. Furthermore, composition of data terms are removed. This removal results in data terms whose pseudo-terms do only have labels and not Xcerpt^{RDF} terms at object positions. As a consequence all subject labels are at the same level which simplifies the referencing of data terms.

The treatment of Xcerpt^{RDF} query terms differs from that of data terms. Here, the consideration for RDF expressivity is unimportant since queries usually provide constructs which are not offered in RDF. For example, variables, optionality and negation which are essential for querying do not have any RDF counterparts. Furthermore, shorthand notations for RDF containers within query terms are enriched by different kinds of brackets used for specifying completeness and order such that these shorthand notations are considered as separate constructs which are not abbreviations any more. Thus, these shorthand notations may not be expanded when occurring within query terms.

4.1 Normalizing Xcerpt^{RDF} Data Terms

When normalizing data terms, shorthand notations are replaced by their respective longhand notations. Furthermore, factorizations, except for the factorization of subjects, are expanded and compositions are resolved. The resulting data terms are of the form $s\{p_1 \rightarrow o_1, \ldots, p_n \rightarrow o_n\}$ such that the o_i are labels and the subjects are all on the same (lowest) level. Thus, root nodes can be omitted. Besides the expansion of the shorthand notations as described in Section 3.3, the normalization consists of several steps which are illustrated in Figure 4.1.

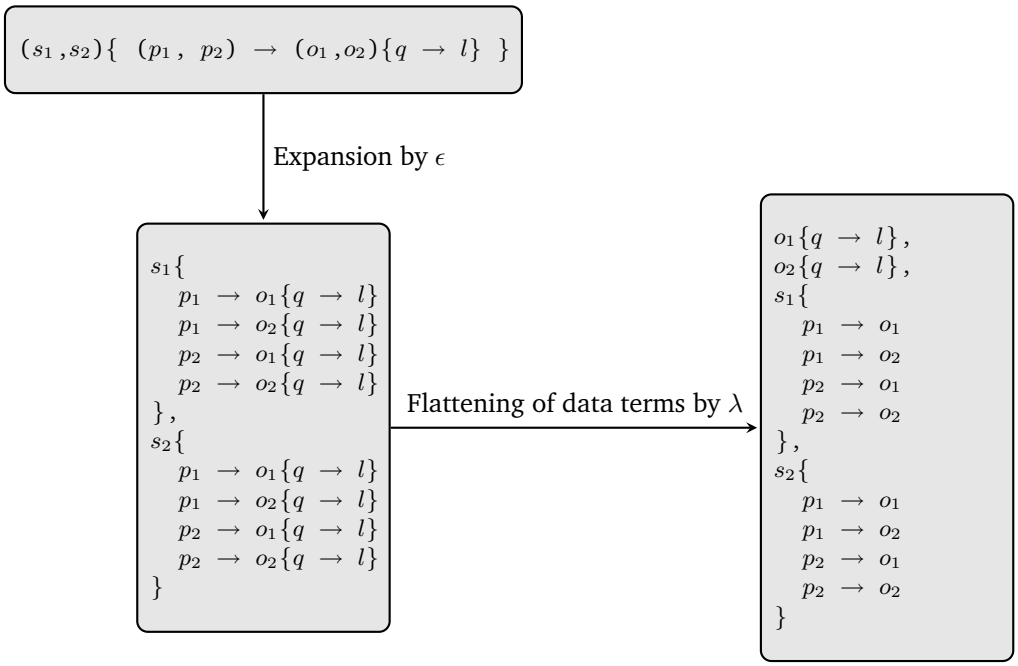


Figure 4.1: Overview of the normalization of an Xcerpt$^{\text{RDF}}$ data term

In the following, each transformation step is explained, where several abbreviations are convenient.

Definition 4.1 (Abbreviations for Lists of Nodes and Data Terms)**.** *Let s denote a URI, blank or literal node which occurs as a subject, let p denote a predicate. Further, let t denote an Xcerpt^{RDF} data term or a URI, blank node or literal node which occurs as an object. A list of subjects $(s_1, s_2, \ldots, s_n)$ will be abbreviated as s^n. The same applies to lists of predicates and Xcerpt^{RDF} data terms.* ✦

With the notation above, an Xcerpt$^{\text{RDF}}$ data term as, for example, $(s_1, s_2)\{(p_1, p_2, p_3) \rightarrow (t_1, t_2)\}$ can be abbreviated as $s^2\{p^3 \rightarrow t^2\}$. A data term like $s\{p_{11} \rightarrow (t_{11}, t_{12}), (p_{21}, p_{22}) \rightarrow t_{21}\}$ can be abbreviated as $s\{p_1^1 \rightarrow t_1^2, p_2^2 \rightarrow t_2^1\}$.

4.1.1 Expanding Xcerpt$^{\text{RDF}}$ Data Terms

The expansion of Xcerpt$^{\text{RDF}}$ data terms removes lists of subjects, predicates and objects by expanding the factorizations in a data term.

Definition 4.2 (Expansion of Xcerpt$^{\text{RDF}}$ Data Terms)**.** *The expansion of Xcerpt^{RDF} data terms is denoted by ϵ and defined as follows.*

1. *If t is a URI, a blank node or a literal in object position then $\epsilon(t) = t$.*

2. $\epsilon\left(s^k\{p_1^{k_1} \to t_1^{l_1}, \ldots, p_n^{k_n} \to t_n^{l_n}\}\right) = \begin{array}{l} \epsilon\left(s_1\{p_1^{k_1} \to t_1^{l_1}, \ldots, p_n^{k_n} \to t_n^{l_n}\}\right), \\ \epsilon\left(s_2\{p_1^{k_1} \to t_1^{l_1}, \ldots, p_n^{k_n} \to t_n^{l_n}\}\right), \\ \vdots \\ \epsilon\left(s_k\{p_1^{k_1} \to t_1^{l_1}, \ldots, p_n^{k_n} \to t_n^{l_n}\}\right) \end{array}$

3. $\epsilon\left(s\{p_1^{k_1} \to t_1^{l_1}, \ldots, p_n^{k_n} \to t_n^{l_n}\}\right) = s\{\epsilon_{st}(p_1^{k_1} \to \epsilon(t_1^{l_1})), \ldots, \epsilon_{st}(p_n^{k_n} \to \epsilon\left(t_n^{l_n}\right))\}$

4. $\epsilon\left(t_1, t_2, \ldots, t_n\right) = \epsilon\left(t_1\right), \epsilon\left(t_2\right), \ldots, \epsilon\left(t_n\right)$.
$\epsilon\left(\left(t_1, t_2, \ldots, t_n\right)\right) = \left(\epsilon\left(t_1\right), \epsilon\left(t_2\right), \ldots, \epsilon\left(t_n\right)\right)$.

The expansion of pseudo-terms is denoted by ϵ_{st} and defined as follows:

4a. $\epsilon_{st}\left(p^i \to t^j\right) = \epsilon_{st}\left(p_1 \to t^j\right), \ldots, \epsilon_{st}\left(p_i \to t^j\right)$

4b. $\epsilon_{st}\left(p \to t^j\right) = p \to t_1, \ldots, p \to t_j$

where $t_1, t_2, \ldots, t_j$ are expanded Xcerpt^{RDF} data terms. ✦

The Example 4.1 illustrates in detail how ϵ is used to expand an Xcerpt$^{\text{RDF}}$ data term.

Example 4.1 (Expansion of Xcerpt$^{\text{RDF}}$ data term)**.**

$$
\begin{array}{rcl}
\epsilon(s_1\{(p_1, p_2) \to ((s_2, s_3)\{q \to l\}, s_4)\}) & \stackrel{3.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (\epsilon((s_2, s_3)\{q \to l\}, s_4))\} \\
& \stackrel{4.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (\epsilon((s_2, s_3)\{q \to l\}), \epsilon(s_4))\} \\
& \stackrel{1.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (\epsilon((s_2, s_3)\{q \to l\}), s_4)\} \\
& \stackrel{2.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (\epsilon(s_2\{q \to l\}), \epsilon(s_3\{q \to l\}), s_4)\} \\
& \stackrel{3.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (s_2\{\epsilon_{st}(q \to \epsilon(l))\}, s_3\{\epsilon_{st}(q \to \epsilon(l))\}, s_4)\} \\
& \stackrel{1.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (s_2\{\epsilon_{st}(q \to l)\}, s_3\{\epsilon_{st}(q \to l)\}, s_4)\} \\
& \stackrel{4b.}{=} & s_1\{\epsilon_{st}((p_1, p_2) \to (s_2\{q \to l\}, s_3\{q \to l\}, s_4))\} \\
& \stackrel{4a.}{=} & s_1\{\epsilon_{st}(p_1 \to (s_2\{q \to l\}, s_3\{q \to l\}, s_4)), \\
& & \quad \epsilon_{st}(p_2 \to (s_2\{q \to l\}, s_3\{q \to l\}, s_4))\} \\
& \stackrel{4b.}{=} & s_1\{p_1 \to s_2\{q \to l\}, p_1 \to s_3\{q \to l\}, p_1 \to s_4, \\
& & \quad p_2 \to s_2\{q \to l\}, p_2 \to s_3\{q \to l\}, p_2 \to s_4\}\text{✦}
\end{array}
$$

It should be mentioned that ϵ_{st} is only defined on $p \to t$, where t is already expanded. That is, ϵ occurrences within an argument of ϵ_{st} must be evaluated first (cf. Example 4.1). This is necessary because otherwise the expansion is incomplete as shown in Example 4.2.

Example 4.2 (Incomplete Expansion by ϵ_{st} in case of t being not already expanded)**.**

$$\begin{aligned}
\epsilon_{st}((p_1, p_2) \rightarrow \epsilon((t_1, t_2))) &\overset{4a.}{=} \epsilon_{st}(p_1 \rightarrow \epsilon((t_1, t_2))), \\
&\quad\ \epsilon_{st}(p_2 \rightarrow \epsilon((t_1, t_2))) \\
&\overset{4b.}{=} p_1 \rightarrow \epsilon((t_1, t_2)), \\
&\quad\ p_2 \rightarrow \epsilon((t_1, t_2)) \\
&\overset{4.}{=} p_1 \rightarrow (\epsilon(t_1), \epsilon(t_2)), \\
&\quad\ p_2 \rightarrow (\epsilon(t_1), \epsilon(t_2)) \\
&= \ldots
\end{aligned}$$

The list $(\epsilon(t_1), \epsilon(t_2))$ will never be eliminated, since the embracing ϵ_{st} is already eliminated before the list of objects is created by ϵ. Thus, the expansion must be computed in a depth-first manner. ✦

In general, the resulting Xcerpt$^{\text{RDF}}$ data term has the form $s\{p_1 \rightarrow t_1, \ldots, p_n \rightarrow t_n\}$, where the t_i are in turn expanded.

The following Proposition 4.1 states the termination of the expansion algorithm.

Proposition 4.1 (Termination of the Expansion of Xcerpt$^{\text{RDF}}$ Data Terms)**.** *The expansion of Xcerpt^{RDF} Data Terms as given in Definition 4.2 terminates.*

Proof. Let an arbitrary data term $t = s^m\{p_1^{k_1} \rightarrow t_1^{l_1}, \ldots, p_n^{k_n} \rightarrow t_n^{l_n}\}$ be given. Recall that the notation $t_n^{k_n}$ is an abbreviation of the list $(t_{n1}, t_{n2}, \ldots, t_{nk_n})$. Assume that $\epsilon(t)$ does not terminate. The termination of ϵ_{st} does not need to be discussed separately for the following reason: since t is finite by assumption, the number of predicates within t is also finite. That is, according to rule 3 the termination of ϵ_{st} only depends on the termination of ϵ. Construct a tree illustrating the incarnations of ϵ on t. The arcs of the tree are thereby labelled by the rules of Definition 4.2 yielding the incarnations of ϵ. The tree is constructed as follows.

- Set $\epsilon(t)$ as the root of the tree.
- Set as children of the root the incarnations of ϵ resulting from the application of rule 2 in Definition 4.2.
- For each resulting data term with only one remaining subject label add as its children the incarnations of ϵ resulting from the application of rule 3.
- Repeat the steps for the resulting incarnations of ϵ for the label or terms at object positions.

The resulting tree is shown in Figure 4.2.

As it can be seen, finitely many steps are necessary to traverse from the subject of t to its objects. We assume that ϵ does not terminate, thus, the tree consists of infinitely many nodes. According to König's Lemma [18], a tree with infinitely many nodes has an infinite branching factor or an infinite branch. That is, either the subject list or the object list of t or some t_{ij} within some of the lists $t_1^{l_1}$ to $t_n^{k_n}$ must be infinite. However, these assertions make t infinite

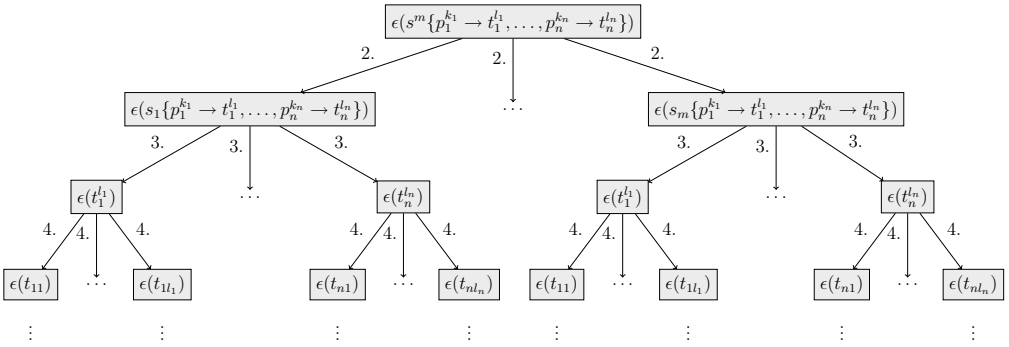


Figure 4.2: Tree illustrating the invocations of ϵ on the data term t. The edges are labelled by the rules of Definition 4.2 yielding the incarnations of ϵ.

which contradicts the assumption that t is finite and, thus, $\epsilon(t)$ terminates. ✦

The rules in Definition 4.2 are selected deterministically, i.e. in each step only one rule is applicable. Thus, the following Proposition 4.2 trivially holds.

Proposition 4.2. *The rules given in Definition 4.2 are confluent.* ✦

4.1.2 Flattening an Expanded Xcerpt$^{\text{RDF}}$ Data Term

As a next step, duplicate occurrences of subjects resulting from the expansion process have to be removed. Furthermore, one has to lower the level of those occurrences to the lowest. This is an important preprocessing step for the forthcoming transformation to Xcerpt$^{\text{XML}}$. In doing so, a global set variable that contains Xcerpt$^{\text{RDF}}$ data terms is used. A set is used because it filters out duplicates by itself. In order to distinguish Xcerpt$^{\text{RDF}}$ data terms from the set, it will be embraced in $\langle\cdot\rangle$. The flattening is defined according to Definition 4.3.

Definition 4.3 (Flattening an Expanded Xcerpt$^{\text{RDF}}$ Data Term)**.** *The flattening including the removal of duplicates of an expanded Xcerpt^{RDF} data term is denoted by λ and defined as follows, where s and o denote URI or blank nodes and δ describes a set of Xcerpt^{RDF} data terms.*

1. $\lambda\,(\; s_1\,\{\, p_{11} \to t_{11}, p_{12} \to t_{12}, \ldots, p_{1n_1} \to t_{1n_1}\},$
   $s_2\,\{\, p_{21} \to t_{21}, p_{22} \to t_{22}, \ldots, p_{2n_2} \to t_{2n_2}\},$
   $\vdots$
   $s_k\,\{\, p_{k1} \to t_{k1}, p_{k2} \to t_{k2}, \ldots, p_{kn_k} \to t_{kn_k}\})\quad \langle\emptyset\rangle \quad =$

   $\lambda\,(\; s_1\,\{\, p_{11} \to t_{11}, p_{12} \to t_{12}, \ldots, p_{1n_1} \to t_{1n_1}\}),$
   $\lambda\,(\; s_2\,\{\, p_{21} \to t_{21}, p_{22} \to t_{22}, \ldots, p_{2n_2} \to t_{2n_2}\}),$
   $\vdots$
   $\lambda\,(\; s_k\,\{\, p_{k1} \to t_{k1}, p_{k2} \to t_{k2}, \ldots, p_{kn_k} \to t_{kn_k}\})\quad \langle\emptyset\rangle$

2. $\lambda\ (\ s_1\ \{\ p_{11} \to t_{11}, p_{12} \to t_{12}, \ldots, p_{1n_1} \to t_{1n_1}\})\quad \langle\delta\rangle \quad =$

$s_1\ \{\ p_{11} \to \lambda(t_{11}), p_{12} \to \lambda(t_{12}), \ldots, p_{1n_1} \to \lambda(t_{1n_1})\}\quad \langle\delta\rangle$

3. $s\ \{\ p_1 \to \lambda(o_1\{u_{11} \to t_{11}, \ldots, u_{1n_1} \to t_{1n_1}\}),$
$p_2 \to \lambda(o_2\{u_{21} \to t_{21}, \ldots, u_{2n_2} \to t_{2n_2}\}),$
$\vdots$
$p_m \to \lambda(o_m\{u_{m1} \to t_{m1}, \ldots, u_{mn_2} \to t_{mn_m}\})\}\quad \langle\delta\rangle \quad =$

$s\{p_1 \to o_1, \ldots, p_m \to o_m\} \quad \langle\delta \cup \{\ o_1\{u_{11} \to t_{11}, \ldots, u_{1n_1} \to t_{1n_1}\},$
$o_2\{u_{21} \to t_{21}, \ldots, u_{2n_2} \to t_{2n_2}\},$
$\vdots$
$o_m\{u_{m1} \to t_{m1}, \ldots, u_{mn_m} \to t_{mn_m}\}\}\rangle$

There, the subjects o_1 to o_m of the term within the resulting set δ are considered to be different. This can be achieved by replacing two data terms $s\{p_{11} \to t_{11}, \ldots, p_{1n_1} \to t_{1n_1}\}$ and $s\{p_{21} \to t_{21}, \ldots, p_{2n_2} \to t_{2n_2}\}$ with identical subjects by the data term resulting from the merge of their pseudo-terms, i.e. $s\{p_{11} \to t_{11}, \ldots, p_{1n_1} \to t_{1n_1}, p_{21} \to t_{21}, \ldots, p_{2n_2} \to t_{2n_2}\}$.

4. $s\{p_1 \to o_1, \ldots, p_n \to o_n\}\quad \langle\{t_1, t_2, \ldots, t_m\}\rangle \quad = \quad \lambda(t_1),$
$\lambda(t_2),$
$\vdots$
$\lambda(t_m),$
$s\{p_1 \to o_1, \ldots, p_n \to o_n\}$
$\langle\emptyset\rangle$

5. *if t is an URI, blank or literal node then $\lambda(t) = t$.* ✦

The following Example 4.3 shows how the flattening works on an expanded Xcerpt$^{\text{RDF}}$ data term.

Example 4.3 (Flattening an Expanded Xcerpt$^{\text{RDF}}$ Data Term)**.**

$$\begin{aligned}
&\lambda(s_1\{p_1 \to s_2\{q \to l\}, p_1 \to s_3\{q \to l\}, p_1 \to s_4\}),\\
&\quad p_2 \to s_2\{q \to l\}, p_2 \to s_3\{q \to l\}, p_2 \to s_4\}) && \langle\emptyset\rangle && \overset{2.}{=}\\
&s_1\{p_1 \to \lambda(s_2\{q \to l\}), p_1 \to \lambda(s_3\{q \to l\}), p_1 \to \lambda(s_4),\\
&\quad p_2 \to \lambda(s_2\{q \to l\}), p_2 \to \lambda(s_3\{q \to l\}), p_2 \to \lambda(s_4)\}\} && \langle\emptyset\rangle && \overset{5.}{=}\\
&s_1\{p_1 \to \lambda(s_2\{q \to l\}), p_1 \to \lambda(s_3\{q \to l\}), p_1 \to s_4,\\
&\quad p_2 \to \lambda(s_2\{q \to l\}), p_2 \to \lambda(s_3\{q \to l\}), p_2 \to s_4\}\} && \langle\emptyset\rangle && \overset{3.}{=}\\
&s_1\{p_1 \to s_2, p_1 \to s_3, p_1 \to s_4,\\
&\quad p_2 \to s_2, p_2 \to s_3, p_2 \to s_4\} && \langle\{s_2\{q \to l\}, s_3\{q \to l\}\}\rangle && \overset{3.}{=}\\
&\lambda(s_2\{q \to l\}),\\
&\lambda(s_3\{q \to l\}),\\
&s_1\{p_1 \to s_2, p_1 \to s_3, p_1 \to s_4,\\
&\quad p_2 \to s_2, p_2 \to s_3, p_2 \to s_4\} && \langle\emptyset\rangle && \overset{2.}{=}\\
&s_2\{q \to \lambda(l)\},\\
&s_3\{q \to \lambda(l)\},\\
&s_1\{p_1 \to s_2, p_1 \to s_3, p_1 \to s_4,\\
&\quad p_2 \to s_2, p_2 \to s_3, p_2 \to s_4\} && \langle\emptyset\rangle && \overset{5.}{=}\\
&s_2\{q \to l\},\\
&s_3\{q \to l\},\\
&s_1\{p_1 \to s_2, p_1 \to s_3, p_1 \to s_4,\\
&\quad p_2 \to s_2, p_2 \to s_3, p_2 \to s_4\} && \langle\emptyset\rangle && \blacklozenge
\end{aligned}$$

As it can be seen in the example above, duplicates of $s_2\{q \to l\}$ and $s_3\{q \to l\}$ are eliminated. Furthermore, all subjects are at the same (lowest) level. In general, the result of flattening is of the form $s_1\{p_{11} \to o_{11}, p_{12} \to o_{12}, \ldots, p_{1n_1} \to o_{1n_1}\}, \ldots, s_k\{p_{k1} \to o_{k1}, p_{k2} \to o_{k2}, \ldots, p_{kn_k} \to o_{kn_k}\}$ where o_{ij} are URI, blank or literal nodes.

The termination of the flattening algorithm can be shown similarly to the termination of the expansion algorithm. The confluence of the flattening rules is also trivially given since only one rule in each rewriting step is applicable.

The factorizations reduce a data term polynomially in space. That is, the normalization of the data term

```
1  (s1, s2, ..., sn) {
2    (p1, p2, ..., pm) → (o1, o2, ..., ok)
3  }
```

results in $O(n)$ data terms each having $O(m \cdot k)$ property-object pseudo-terms.

Chapter 5

Evaluation of Queries and Data in Xcerpt^{RDF}

The evaluation in Xcerpt^{RDF} uses the notion of simulation of Xcerpt^{RDF} terms into Xcerpt^{RDF} terms. Using simulation for this purpose has been introduced in [22]. Intuitively, asking whether an Xcerpt^{RDF} term t simulates into an Xcerpt^{RDF} term t' is equal to asking whether t can be found in t' with respect to the resources involved as well as the structure of t. For example, _:X {eg:knows → eg:john} simulates in eg:anna { eg:knows → eg:john, eg:name → "Anna"} because the first term can be found in the latter term assuming that _:X is eg:anna which is admissible since the blank node _:X can be seen as an existential quantified variable. Additional constraints (regarding completeness and order) can be stipulated which is done in Xcerpt^{RDF} by several single or double brackets.

Due to RDF and Xcerpt^{RDF} specifities, the simulation of Xcerpt^{XML} must be adapted to RDF. In contrast to Xcerpt^{XML}, the simulation in Xcerpt^{RDF} has no explicit root since RDF graphs (and thus their syntactic representations) have no roots. In query terms the shorthand notations of RDF containers are retained and, thus, simulation needs to be extended in order to deal with these shorthand notation. Regarding RDF, the data model is completely different from that of XML: There are directed labelled edges, and occurrences of labels are not distinguished. Blank nodes must also be considered. Among other things, that is why the matching-relation (cf. Section 5.1.1) between labels must be a many-to-one relationship (i.e. a partial function).

The syntax of Xcerpt^{RDF} offers a great latitude in writing queries and RDF data. Hence, in order to define a clear cut evaluation mechanism it is rational to restrict the set of Xcerpt^{RDF} terms which are evaluated. The remaining Xcerpt^{RDF} terms need to be normalized according to Section 4. Moreover, several language constructs discussed in this thesis are omitted in the evaluation. This results from the lack of time and from the decision to introduce the simulation on a sufficient but still clear subset of Xcerpt^{RDF}. The concepts which are disregarded in the evaluation method of Xcerpt^{RDF} are described below.

- Variables for Concise Bounded Descriptions
- Simulation of Construct Terms into Query Terms

- Substitution and Substitution Sets
- Named Graphs
- Filtering answers using WHERE-clauses of Xcerpt^{XML}

None of the well-known RDF query languages follow the approach of a simulation based evaluation of RDF queries and data. For example, SPARQL is based on an algebraic semantics and TRIPLE is based on a model-theoretic semantics. But the following sections illustrate that the concept of simulation of Xcerpt^{RDF} terms in Xcerpt^{RDF} terms is a perfectly appropriate means to evaluate RDF data. So, the computation of answers for queries as a key issue is based on the simulation of query terms in data terms. As an optimization of query evaluation subsumption is defined by the simulation of query terms into data terms. Finally, leanness of RDF graphs can be checked by the simulation of data terms into data terms.

5.1 Preliminaries

The discussion of Xcerpt^{RDF}'s semantics makes use of the following notations.

Definition 5.1 (Sets of Labels and Xcerpt^{RDF} Terms)**.** *In order to specify the semantics of Xcerpt^{RDF} the following sets are useful.*

- U *denotes the set of URI labels.*
- B *denotes the set of blank nodes.*
- L *denotes the set of literals.*
- $N = U \cup B \cup L$ *denotes the set of all labels (or RDF nodes).*
- V *denotes the set of Xcerpt^{RDF} variables.*
- X *denotes the set of Xcerpt^{RDF} terms.*
- $Q \subseteq X$ *denotes the set of Xcerpt^{RDF} query terms.*
- $Q_g \subseteq Q$ *denotes the set of Xcerpt^{RDF} ground query terms.*
- $D \subseteq Q_g$ *denotes an arbitrary set of normalized Xcerpt^{RDF} data terms.* ✦

Moreover, in the discussion of the evaluation method it is often referred to the slots (i.e. property-object pairs) and container shorthand notations within an Xcerpt^{RDF} term. To simplify matters, these components are named as given in Definition 5.2.

Definition 5.2 (Pseudo-Term)**.** *The property object-pairs and container shorthand notations appearing within the brackets of a given Xcerpt^{RDF} term are denoted as pseudo-terms. More specifically,*

- *a property-object pair of the form $p \rightarrow o$ is called a positive property-object pseudo-term.*

- *a property-object pair of the form (without p → o) is called a negative property-object pseudo-term.*
- *a property-object pair of the form (optional p → o) is called an optional property-object pseudo-term.*
- *the bagOf, seqOf, altOf shorthand notations are called container pseudo-terms. They are called negative (optional, resp.) if they occur within the scope of without (optional, resp.).*
- *the listOf shorthand notation is called an collection pseudo-term. It is called negative (optional, resp.) if it occurs within the scope of without (optional, resp.).* ✦

Note that the evaluation of optional and negative property-object pseudo-terms is restricted to those which are of the form (optional p → o) and (without p → o), respectively. In order to simplify matters, the remaining scopes which are given in Example 3.2 are disregarded. Besides, query terms which contain negative or optional property-object pseudo-terms with other scopes can be rewritten to queries that do not contain pseudo-terms with such scopes. However, these rewritings often necessitate WHERE-clauses and can be very involved. For example, the query term $a\{\{(without\ p) \rightarrow o\}\}$ is equivalent to the query $a\{\{q \rightarrow o\}\}$WHERE$q \neq p$. As another example, the query term $s\{\{p \rightarrow (without\ o\{\{q \rightarrow l\}\})\}\}$ can be rewritten to the following query:

```
1 or {
2  and {
3     s {{p → o}},
4     not o {{q → l}}
5  },
6  s {{p → u}}
7 }
8 WHERE u ≠ o
```

However, it is suggested to extend the evaluation of Xcerpt$^{\text{RDF}}$ queries in forthcoming versions to also support the remaining scopes of negative and optional property-object pseudo-terms which are given in Example 3.2.

Furthermore, the simulation uses various subsets of the set of pseudo-terms of a query term. These (sub-)sets are given in Definition 5.3.

Definition 5.3 (Set of Pseudo-Terms of a Query Term and its Partitioning)**.** *For an arbitrary Xcerpt^{RDF} term t*

- *$PT(t)$ denotes the set of all pseudo-terms of t.*
- *$PT^{+}(t)$ denotes the set of positive pseudo-terms of t, i.e. pseudo-terms preceeded neither by $without$ nor $optional$.*
- *$PT^{-}(t)$ denotes the set of negative pseudo-terms of t, i.e. pseudo-terms preceeded by $without$.*
- *$PT^{?}(t)$ denotes the set of optional pseudo-terms of t, i.e. pseudo-terms preceeded by $optional$.*

- $PT_{cont}(t)$ *denotes the set of (positive, optional and negative) container pseudo-terms. Respective partitions are indicated by* $PT^{+}_{cont}(t)$ *(for the set of positive container pseudo-terms),* $PT^{?}_{cont}(t)$ *(for the set of optional container pseudo-terms) and by* $PT^{-}_{cont}(t)$ *(for the set of negative container pseudo-terms).*
- $PT_{PO}(t)$ *denotes the set of (positive, optional and negative) property-object pseudo-terms. Respective parititions are indicated by* $PT^{+}_{PO}(t)$ *(for the set of positive property-object pseudo-terms),* $PT^{?}_{PO}(t)$ *(for the set of optional property-object pseudo-terms) and by* $PT^{-}_{PO}(t)$ *(for the set of negative property-object pseudo-terms)* ✦

The Figure 5.1 illustrates one kind of partitioning of a set of query pseudo-terms which is often used in this thesis. Here, a set within the the tree consists of its children.

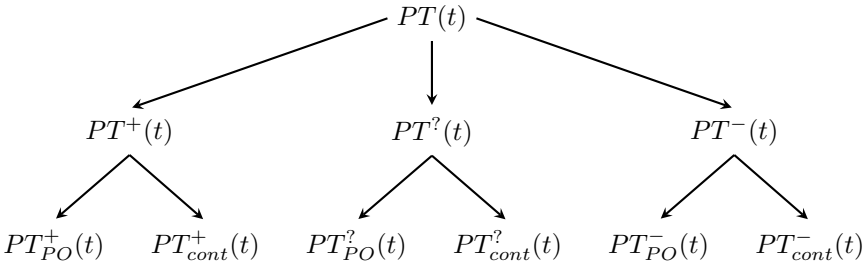


Figure 5.1: The partitioning of an arbitrary set of query pseudo-terms. The naming of the sets is according to Defintition 5.3. Read the directed arcs as "consists of". Siblings in the tree are supposed to be pairwise disjunct. The relationships $PT_{PO}(t) = PT^{+}_{PO}(t) \cup PT^{?}_{PO}(t) \cup PT^{-}_{PO}(t)$ and $PT_{cont}(t) = PT^{+}_{cont}(t) \cup PT^{?}_{cont}(t) \cup PT^{-}_{cont}(t)$ are not shown in the figure.

Moreover, it is useful to provide a function called Sub which returns the subject of a given Xcerpt$^{\text{RDF}}$ term and functions $Prop$ and Obj which return the property and object of a given property-object pseudo-term. These functions allow to access the labels of property-object pseudo-terms indepedently from their actual kinds (positive, optional or negative).

Definition 5.4 (Label-Projections of Xcerpt$^{\text{RDF}}$ Terms)**.** *Let* t *denote an Xcerpt^{RDF} term. Define the following label-projections.*

- $Sub(t)$ *is the subject of* t*.*
- *For* $st \in PT_{PO}(t)$ *which may be of the form* $p \rightarrow q$*, (without* $p \rightarrow q$*) or (optional* $p \rightarrow q$*)*
  - $Prop(st) = p$
  - $Obj(st) = q$ ✦

5.1.1 Matching Labels

The simulation of Xcerpt$^{\text{RDF}}$ terms uses the notion of matching labels. Testing whether a label matches against another means testing whether both labels denote the same resource. Thus,

matching is weaker than testing equality since equality of both labels is generally not required for matching. For example, two different blank nodes are supposed to match since it cannot be concluded that two different blank nodes also denote two different resources.

Definition 5.5 (Matching of Labels)**.** *A label l_1 matches against another label l_2, written as $l_1 \trianglerighteq l_2$, if one of the following cases hold.*

- *l_1 and l_2 are both literals and $l_1 = l_2$.*
- *l_1 and l_2 are URIs (or expanded qualified names) and $l_1 = l_2$.*
- *l_1 is a blank node and l_2 is a URI.*
- *l_1 and l_2 are blank nodes.*
- *l_1 is a regular expression, l_2 is a label and $l_2 \in \mathcal{L}(l_1)$.*
- *l_1 and l_2 are regular expressions and $\mathcal{L}(l_1) \supseteq \mathcal{L}(l_2)$.*

For a given regular expression (or string) r, $\mathcal{L}(r)$ denotes the language expressed by r. ✦

The matching of labels is generally not symmetric and is directed from the more generic to the more specific case.

Matching of labels is also used to retrieve data terms from a known dataset according to their subjects. That is, for a given label there is a data term to be found within the dataset such that the label matches against the subject of that data term. The function that accomplishes this task is called data term selector and is defined as follows.

Definition 5.6 (Data Term Selector)**.** *Let $G \subseteq D$ be a set of normalized Xcerpt^{RDF} data terms, then define $\lambda_G \colon N \to G$ which maps a label l to a data term $\lambda_G(l)$ such that $l \trianglerighteq Sub(\lambda_G(l))$.* ✦

λ_G can be naively implemented by traversing the list of data terms and trying to match the label against each subject. If such a subject is found then λ returns the entire data term.

Note that data terms are normalized, thus, subjects are all at the same level whereas query terms remain composed. The advantages of this approach is discussed in Section 5.4.2 after the simulation of ground query terms in data terms is introduced.

5.2 Answering Queries - Simulation of Ground Query Terms into Data Terms

The simulation of ground query terms into data terms is used in order to find answers for Xcerpt^{RDF} queries by determining whether a given substitution is valid. This can be pictured as follows. First, a substitution is guessed which at least makes the subject and all positive and negative pseudo-terms of a given query term ground. Next, the validity of the substitution is verified by simulating the ground query term into the data terms. The guessed substitution is rejected, if the simulation fails.

While guessing of suitable substitution simplifies the discussion of the simulation of ground query terms, operationally, substitutions are determined by simulation unification [22], a unification algorithm perfectly appropriate for graph-structured data models which is based on the simulation of graphs.

In the simulation of ground query terms into data terms, optional pseudo-terms play a special role. If a guessed substitution provides bindings for variables occurring within an optional pseudo-term making the optional pseudo-term ground, this pseudo-term has to be treated as an ordinary positive pseudo-term. Otherwise, if the optional pseudo-term is not made ground by the substitution to be verified, the pseudo-term is deleted from the set of the query pseudo-terms and hence not considered in the simulation. As a consequence, optional pseudo-terms does not need to be separately considered.

The Definition 5.8 introduces the simulation of ground query terms into data terms, for now disregarding container pseudo-terms which are added to the simulation later. Thereby, the notion of pseudo-term simulation as given in Definition 5.9 is used. In the simulation, the existence of a finite many-to-one relation $\mathcal{R}_{\trianglerighteq}$ is assumed which fulfills the property that for all label-pairs $(x, y) \in \mathcal{R}_{\trianglerighteq}$ it holds that $x \trianglerighteq y$. For a simulation, $\mathcal{R}_{\trianglerighteq}$ must exist and provide appropriate matching-partners for the simulation to succeed. Operationally, this is achieved by constructing $\mathcal{R}_{\trianglerighteq}$ during the simulation, thereby testing the restrictions of $\mathcal{R}_{\trianglerighteq}$ given in Definition 5.7.

Definition 5.7 (Finite Matching-Relation)**.** *Define the finite matching relation $\mathcal{R}_{\trianglerighteq} \subseteq N \times N$ such that*

1. *$\mathcal{R}_{\trianglerighteq}$ satisfies the many-to-one property, i.e. for all $(x, y) \in \mathcal{R}_{\trianglerighteq}$ it holds that if $(x, y) \in \mathcal{R}_{\trianglerighteq}$ and $(x, z) \in \mathcal{R}_{\trianglerighteq}$ then $y = z$.*
2. *$\mathcal{R}_{\trianglerighteq} \neq \emptyset$*
3. *For all $(x, y) \in \mathcal{R}_{\trianglerighteq}$ it holds that $x \trianglerighteq y$.* ✦

With the finite matching relation $\mathcal{R}_{\trianglerighteq}$ the simulation of ground query terms can be defined as given in Definition 5.8.

Definition 5.8 (Simulation of Ground Query Terms in Data Terms)**.** *Let $q \in Q_g, d \in D' \subseteq D$. One says that q simulates (via $\mathcal{R}_{\trianglerighteq}$) in d which is written $q \preceq_{\mathcal{R}_{\trianglerighteq}} d$ if $(Sub(q), Sub(d)) \in \mathcal{R}_{\trianglerighteq}$ and the following holds.*

1. *Case: q does not contain negative pseudo-terms, i.e. $PT^-(q) = \emptyset$.*
   An injective and total mapping $\pi\colon PT^+(q) \to PT(d)$ exists such that for all $s \in PT^+(q)$ it holds that $s \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \pi(s)$.
2. *Case: q contains negative pseudo-terms, i.e. $PT^-(q) \neq \emptyset$.*
   (a) *An injective and total mapping $\pi\colon PT^+(q) \to PT(d)$ exists with the properties as defined above.*
   (b) *π cannot be extended to a $\overline{\pi}\colon (PT^+(q) \cup PT^-(q)) \to PT(d)$ such that for some $s^- \in PT^-(q)$ it holds that $s^- \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \overline{\pi}(s^-)$.* ✦

The Definition 5.8 uses the simulation of query pseudo-terms in data pseudo-terms which is defined in Definition 5.9.

Definition 5.9 (Simulation of Query Pseudo-Terms in Data Pseudo-Terms)**.** *Let a query pseudo-term s_q of a query term q and and a data pseudo-term s_d of a data term $d \in D' \subseteq D$ be given. One says that s_q simulates via $\mathcal{R}_{\trianglerighteq}$ in s_d which is written $s_q \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} s_d$ if following holds.*

1. $(Prop(s_q), Prop(s_d)) \in \mathcal{R}_{\trianglerighteq}$
2. *If* $Obj(s_q) \in N$ *then* $(Obj(s_q), Obj(s_d)) \in \mathcal{R}_{\trianglerighteq}$
3. *If* $Obj(s_q) \in Q_g$ *then* $Obj(s_q) \preceq_{\mathcal{R}_{\trianglerighteq}} \lambda_{D'}(Obj(s_d))$.

Definition 5.8 shows that the simulation of ground query terms into data terms is basically the same as introduced in [22] except the modifications which are necessary due to the differences between Xcerpt$^{\text{RDF}}$ and Xcerpt$^{\text{XML}}$. For example, the relation $\mathcal{R}_{\trianglerighteq}$ basically relates blank nodes to other blank nodes. In cases of other labels, $\mathcal{R}_{\trianglerighteq}$ is the identity. That is, the relation $\mathcal{R}_{\trianglerighteq}$ is not needed in Xcerpt$^{\text{XML}}$ since blank nodes do not occur in Xcerpt$^{\text{XML}}$.

The following Example 5.1 illustrates the simulation of ground query terms into data terms.

Example 5.1 (Simulation of Ground Query Terms into Data Terms)**.** *Let the following set of normalized data terms D' be given.*

1 $u \ \{ \ v \ \rightarrow \ w \ \} \quad (d_1)$
2 $w \ \{ \ x \ \rightarrow \ y \ \} \quad (d_2)$

Let the ground query term q be $a\{\{b \rightarrow c\{\{d \rightarrow e\}\}\ \}\}$. Let $\mathcal{R}_{\trianglerighteq} = \{(a,u),(b,v),(c,w),(d,x),(e,y)\}$. Then, $q \preceq_{\mathcal{R}_{\trianglerighteq}} d_1$ because:

1. $(Sub(q), Sub(d_1)) = (a,u) \in \mathcal{R}_{\trianglerighteq}$
2. $\pi(b \rightarrow c\{\{d \rightarrow e\}\}) = v \rightarrow w$ *and* $b \rightarrow c\{\{d \rightarrow e\}\} \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} v \rightarrow w$ *because*
   - $(b,v) \in \mathcal{R}_{\trianglerighteq}$ *and*
   - $(c,w) \in \mathcal{R}_{\trianglerighteq}$. *It remains to show that* $c\{\{d \rightarrow e\}\} \preceq_{\mathcal{R}_{\trianglerighteq}} \lambda_{D'}(w)$ *with* $\lambda_{D'}(w) = d_2$.
3. $c\{\{d \rightarrow e\}\} \preceq_{\mathcal{R}_{\trianglerighteq}} d_2$ *because* $(c,w) \in \mathcal{R}_{\trianglerighteq}$ *and* $d \rightarrow e \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \pi(d \rightarrow e)$ *with* $\pi(d \rightarrow e) = x \rightarrow y$ *because* $(d,x) \in \mathcal{R}_{\trianglerighteq}$ *and* $(e,y) \in \mathcal{R}_{\trianglerighteq}$.

Suppose, that q were $a\{\{b \rightarrow c\{\{(without\ d \ \rightarrow \ e)\}\}\ \}\}$. Then q would not simulate into d_1 anymore. The beginning of the simulation is the same as described above. However, now π may not be extensible to a $\overline{\pi}$ such that the negative pseudo-term would simulate. However, point 3 of the derivation above shows that π is extensible such that $(without\ d \ \rightarrow \ e) \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} x \rightarrow y$. Thus, q does not simulate into d_1. ✦

As next, the simulation-relations as given in the Definitions 5.8 and 5.9 are extended with capabilities to cope with container subterms.

5.3 The Simulation of Query Terms with Containers in Data Terms

In the last section it has been shown how simulation basically works. In order to simplify matters, container pseudo-terms have been omitted in the simulation (cf. Definitions 5.12 and 5.13). This section extends the simulation as discussed in Section 5.2 with capabilities to also evaluate container pseudo-terms.Therefore, several definitions are introduced in the following section.

5.3.1 Preliminaries for the Evaluation of Container Pseudo-Terms

As mentioned in Section 3.4.2, shorthand notations for RDF containers cannot be expanded to RDF predicate-object-pairs, since the various kinds of bracketing the elements cannot be expressed in RDF. As a consequence, the evaluation of these shorthand notations must be defined separately. Therefore, several definitions are convenient.

Definition 5.10. *Let the set of all Xcerpt^{RDF} containers be denoted as PT_{cont}. Let an arbitrary Xcerpt^{RDF} container $c \in PT_{cont}$ be given. For an arbitrary data term t define the mapping $\mathcal{U}\colon PT_{cont}(t) \to U$ such that*

$$\mathcal{U}(c) = \begin{cases} \text{rdf:Bag} & \text{if c is a bagOf pseudo-term} \\ \text{rdf:Alt} & \text{if c is an altOf pseudo-term} \\ \text{rdf:Seq} & \text{if c is a seqOf pseudo-term} \end{cases}$$

Furthermore, let the (multi-)set of the elements of c be denoted as $\mathcal{E}(c)$.
A container element $e \in \mathcal{E}(c)$ of the form $pos\ N\ o$ is called extended and $index(e)$ is the integer number represented by N is called its position.
A container element of the form e is called unextended and $index(e)$ is the position of e within the element list.
For an RDF container membership description $m \in PT$ of the form rdf:_N $\to o$ *(N is a numeric literal) $index(m)$ is the integer number represented by N, otherwise $index(m)$ is undefined.* ✦

The evaluation of container pseudo-terms uses the notion of suitable mappings for container pseudo-terms. These are mappings which adhere the specification of container elements (indicated by the brackets used in the container pseudo-term).

Definition 5.11 (Suitable Mapping for a Container Subterm)**.** *Let a query term q be given and let $c \in PT_{cont}(q)$ be a container pseudo-term of q. Furthermore, let be an arbitrary data term d be given. The mapping $\gamma\colon \mathcal{E}(c) \to PT(d)$ is called suitable for c if it meets the following conditions.*

1. *If the elements of c are unextended, i.e. of the form e then the following holds.*
   (a) *If the elements are specified as unordered and complete (indicated by the brackets $\{\cdot\}$) then γ is element-bijective.*
   (b) *If the elements are specified as unordered and incomplete (indicated by the brackets $\{\{\cdot\}\}$) then γ is injective.*
   (c) *If the elements are specified as ordered and complete (indicated by the brackets $[\cdot]$) then γ is element-bijective and position-preserving.*

(d) *If the elements are specified as ordered and incomplete (indicated by the brackets $[[\cdot]]$) then γ is injective and position-monotone.*

2. *If c contains extended elements, i.e. elements which are of the form $pos\ N\ o$ then following holds.*

   (a) *If the elements are specified as complete (indicated by brackets $\{\cdot\}$ or $[\cdot]$) then γ is element-bijective and position-preserving.*

   (b) *If the brackets are specified as incomplete (indicated by the brackets $\{\{\cdot\}\}$ or $[[\cdot]]$) then γ is injective and position-preserving.*

There, γ is called

- *injective, if and only if for all $x, y \in \mathcal{E}(c)$ it holds that whenever $\gamma(x) = \gamma(y)$ then $x = y$.*
- *element-bijective, if and only if γ is injective and for all pseudo-terms $s \in PT(d)$ describing a container membership an $x \in \mathcal{E}(c)$ exists such that $\gamma(x) = s$.*
- *position-preserving, if and only if for all $x \in \mathcal{E}(c)$ it holds that $index(x) = index(\gamma(x))$.*
- *position-monotone, if and only if for all $x, y \in \mathcal{E}(c)$ it holds that if $index(x) < index(y)$ then $index(\gamma(x)) < index(\gamma(y))$.* ✦

5.3.2 Extending the Simulation of Ground Query Terms in Data Terms

The extension of the simulation relation for container pseudo-terms introduces a certain amount of complexity. This results from the fact that a single container pseudo-terms simulates into several property-object pseudo-terms. The brackets used to parenthesize the elements of the container also have meanings (cf. Section 3.4.2). Moreover, the elements themselves may again be positive, optional or negative. As a consequence, the evaluation of a container pseudo-term yields a separate simulation embedded in the simulation of ground query terms in data terms. This embedded simulation, however, is very similar to the simulation as introduced in the Section 5.2. Consider the following Definition 5.12.

Definition 5.12 (Simulation of Ground Query Terms with Container Pseudo-Terms in Data Terms)**.** *Let $q \in Q_g, d \in D' \subseteq D$. One says that q simulates (via $\mathcal{R}_{\trianglerighteq}$) in d which is written $q \preceq_{\mathcal{R}_{\trianglerighteq}} d$ if $(Sub(q), Sub(d)) \in \mathcal{R}_{\trianglerighteq}$ and the following holds.*

1. *Case: q does not contain negative pseudo-terms, i.e. $PT^-(q) = \emptyset$.*

   (a) *An injective and total mapping $\pi\colon PT^+(q) \to PT(d)$ exists such that for all $s \in PT^+(q)$ it holds that $s \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \pi(s)$.*

   (b) *For each $c_i \in PT^+_{cont}(q)$ there exists a total mapping $\gamma_i\colon \mathcal{E}^+(c_i) \to PT(d)$ suitable for c_i with the following properties:*

   - *γ_i is total on $\mathcal{E}^+(c_i)$ and for all container elements e^+ of the form $pos\ N\ e$ or e it holds that $e^+ \preceq^{\text{elem}}_{\mathcal{R}_{\trianglerighteq}} \gamma_i(e^+)$.*
   - *γ_i cannot be extended to a $\overline{\gamma_i}\colon (\mathcal{E}^+(c_i) \cup \mathcal{E}^-(q)) \to PT(d)$ such that for some negative container element e^- of the form $(without\ e)$ it holds that $e \preceq^{\text{elem}}_{\mathcal{R}_{\trianglerighteq}} \overline{\gamma_i}(e)$.*

(c) *The images of the γ_i are pairwise disjoint.*

(d) *Let* $\text{Im}_\gamma = \bigcup \text{Image}(\gamma_i)$. *Then for all* $s_{\text{PO}} \in PT_{\text{PO}}(q)$ *it holds that* $\pi(s_{\text{PO}}) \in PT(d) \setminus \text{Im}_\gamma$

(e) $\text{Image}(\pi) \cap \text{Im}_\gamma = \emptyset$.

2. *Case:* q *contains negative pseudo-terms, i.e.* $PT^-(q) \neq \emptyset$.

   (a) *An injective and total mapping* $\pi\colon PT^+(q) \to PT(d)$ *exists with the properties as defined above.*

   (b) π *cannot be extended to a* $\overline{\pi}\colon (PT^+(q) \cup PT^-(q)) \to PT(d)$ *such that for some* $s^- \in PT^-(q)$ *it holds that* $s^- \preceq^{\text{pt}}_{\mathcal{R}_\trianglerighteq} \overline{\pi}(s^-)$ *and such that following properties are satisfied.*

      i. *For some* $c^- \in PT^-_{cont}(q)$ *a total and injective mapping* $\gamma_i^-\colon \mathcal{E}^+(c_i^-) \to PT(d)$ *which is suitable for* c_i^- *exists with the following properties.*

         - *For all container elements* e^+, *i.e. elements which are of the form* $pos\ N\ e$ *or* e, γ_i^- *is defined and it holds that* $e^+ \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} \gamma_i^-(e^+)$.
         - γ_i^- *cannot be extended to a* $\overline{\gamma_i^-}\colon (\mathcal{E}^+(c_i) \cup \mathcal{E}^-(c_i)) \to PT(d)$ *such that for some negative container element* e^- *which is of the form* $(without\ e)$ *it holds that* $e^- \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} \overline{\gamma_i^-}(e^-)$.

      ii. *The images of the* γ_i^- *are pairwise disjoint.*

      iii. *With* $\text{Im}_{\gamma^-} = \bigcup \text{Image}(\gamma_i^-)$. *for all* $s^-_{\text{PO}} \in PT^-_{\text{PO}}(q)$ *it is* $\overline{\pi}(s^-_{\text{PO}}) \in PT(d) \setminus \text{Im}_{\gamma^-}$

      iv. $\text{Image}(\overline{\pi}) \cap \text{Im}_{\gamma^-} = \emptyset$. ✦

Definition 5.12 uses the simulation of query pseudo-terms in data pseudo-terms which is defined in Definition 5.13.

Definition 5.13 (Simulation of Query Pseudo-Terms in Data Pseudo-Terms)**.** *Let a query pseudo-term* s_q *of a query term* q *and and a data pseudo-term* s_d *of a data term* $d \in D' \subseteq D$ *be given.* s_q *simulates (via* $\mathcal{R}_\trianglerighteq$*) in* s_d *which is written* $s_q \preceq^{\text{pt}}_{\mathcal{R}_\trianglerighteq} s_d$ *if following holds.*

- *If* $s_q \notin PT_{cont}(q)$ *then*

  1. $(Prop(s_q), Prop(s_d)) \in \mathcal{R}_\trianglerighteq$
  2. *If* $Obj(s_q) \in N$ *then* $(Obj(s_q), Obj(s_d)) \in \mathcal{R}_\trianglerighteq$
  3. *If* $Obj(s_q) \in Q_g$ *then* $Obj(s_q) \preceq_{\mathcal{R}_\trianglerighteq} \lambda_{D'}(Obj(s_d))$.

- *If* $s_q \in PT_{cont}(q)$ *then*

  1. $(Prop(s_d), \text{rdf:type}) \in \mathcal{R}_\trianglerighteq$,
  2. $(\mathcal{U}(s_q), Obj(s_d)) \in \mathcal{R}_\trianglerighteq$. ✦

Finally, the definition uses the simulation of container elements in data pseudo-terms which is defined in Definition 5.14.

Definition 5.14 (Simulation of Container Elements in a Data Subterm)**.** *Let a ground container element* e *of the form* $pos\ N\ e'$ *or of the form* e' *be given. Furthermore, let a data pseudo-term* $s \in PT(d)$ *of a data term* $d \in D' \subseteq D$ *be given.* e *simulates (via* $\mathcal{R}_\trianglerighteq$*) in* s *which is written* $c \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} s$ *if the following holds.*

1. *s describes a container membership, i.e. is of the form rdf:_N $\rightarrow$ o, where N is a numeric literal.*
2. *If $e' \in N$ then $(e', Obj(s)) \in \mathcal{R}_{\trianglerighteq}$.*
3. *If $e' \in Q_g$ then $e' \preceq_{\mathcal{R}_{\trianglerighteq}} \lambda_{D'}(Obj(s))$.* ✦

Consider Example 5.2 below which illustrates the simulation of container pseudo-terms in data pseudo-terms.

Example 5.2 (Simulation of Container Pseudo-Terms in Data Pseudo-Terms)**.** *Let the data term d be*

```
eg:cart {
  rdf:type   → rdf:Bag
  rdf:_1     → eg:milk
  rdf:_2     → eg:coffee
  rdf:_3     → eg:sugar
  eg:weight  → "2.5kg"
}
```

and let the query term q be

```
_:whatever {{
  bagOf{{ _:item1 , _:item2 }}
  (optional rdf:_2 → eg:coffee)
  (without eg:currency → eg:euro)
}}
```

Since the optional pseudo-term of q is ground it will be treated as an ordinary positive pseudo-term. Hence, the ground query term to be simulated in the data term is the following.

```
_:whatever {{
  bagOf{{ _:item1 , _:item2 }}
  rdf:_2 → eg:coffee
  (without eg:currency → eg:euro)
}}
```

Let $\mathcal{R}_{\trianglerighteq} = \{$(_:whatever,eg:cart), (rdf:Bag,rdf:Bag), (_:item1,eg:milk), (_:item2,eg:sugar)$\}$. Then, following holds.

- *(_:whatever,eg:cart) $\in \mathcal{R}_{\trianglerighteq}$.*
- *bagOf{{_:item1, _:item2}} $\preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}}$ rdf:type $\rightarrow$ rdf:Bag because $\mathcal{U}$(bagOf{{_:item1, _:item2}}) = rdf:Bag and (rdf:Bag,rdf:Bag) $\in \mathcal{R}_{\trianglerighteq}$.*
- *For $\mathcal{R}_{\trianglerighteq}$ an injective (thus suitable for bagOf{{_:item1, _:item2}}) mapping γ exists:*
  - *γ(_:item1) =rdf:_1 $\rightarrow$ eg:milk, _:item1 $\preceq^{\text{elem}}_{\mathcal{R}_{\trianglerighteq}}$ eg:milk because (_:item1, eg:milk) $\in \mathcal{R}_{\trianglerighteq}$.*
  - *γ(_:item2) =rdf:_3 $\rightarrow$ eg:sugar, _:item2 $\preceq^{\text{elem}}_{\mathcal{R}_{\trianglerighteq}}$ eg:sugar because (_:item2, eg:sugar) $\in \mathcal{R}_{\trianglerighteq}$.*

$\mathrm{Im}_\gamma = \mathrm{Image}(\gamma) = \{$ *rdf:_1 → eg:milk, rdf:_3 → eg:sugar* $\}$. *Thus, for the simulation of non-container pseudo-terms of* q *the following set of data pseudo-terms are available for the remaining pseudo-term simulation:* $\{$*rdf:_2 → eg: coffee, eg:weight → "2.5kg"* $\}$. *This set contains the data pseudo-term rdf:_2 → eg: coffee in which the query pseudo-term rdf:_2 → eg: coffee simulates. Since for the remaining query pseudo-term (without eg:currency → eg:euro) there exists no simulation partner (because eg:currency → eg:euro does not simulate in the remaining data pseudo-term eg:weight → "2.5kg") it holds that* q *simulates in* d.

There are six different extensions of $\mathcal{R}_{\trianglerighteq}$ *and for each of these extensions a suitable mapping for the bag pseudo-term exists.*

Suppose another query q' *of the form*

```
1  _:whatever {{
2    bagOf{{ pos 1 _:item1, pos 2 _:item2 }}
3  }}
```

Identical to the simulation of q *in* d, $($*_:whatever,eg:cart*$) \in \mathcal{R}_{\trianglerighteq}$. *For the container pseudo-term which is very similar to the container pseudo-term in* q, *except the fact that the container elements are extended by position specifications, only one extension of* $\mathcal{R}_{\trianglerighteq}$ *exists such that there is a suitable mapping* γ *which maps pos 1 _:item1 to rdf:_1 → eg:milk and pos 2 _:item2 rdf:_2 → eg: coffee. Thus,* q' *also simulates in* d, *but from the six possibilities of simulating the container elements of* q *in* d *are only one remains for the container elements in* q'. ✦

5.4 Notes on the Simulation of Ground Query Terms in Data Terms

During the investigation of the simulation of ground query terms in data terms, several interesting topics came up. These are discussed in the following section.

5.4.1 Simulation into Sets of Normalized Data Terms, Connectives for Query Terms

A ground query term may simulate in several or all data terms of a given set of normalized data terms. As mentioned above, this results from the fact that the data terms are normalized, thus, the subjects are all on the same level, whereas in query terms subjects may occur on abitrary levels. Unlike in Xcerpt$^{\text{XML}}$, the structure of the query term, which results from composition, is irrelevant. Thus, a ground query term can also be said to simulate in a set of normalized data terms which can be defined as follows.

Definition 5.15 (Simulation of a Ground Query Term in a Set of Normalized Data Terms)**.** *Let* $D' \subseteq D$ *be a set of normalized data terms and* t_q *be a ground query term.* t_q *simulates (via* $\mathcal{R}_{\trianglerighteq}$*) in* D', *which is written as* $t_q \preceq_{\mathcal{R}_{\trianglerighteq}} D'$, *if a* $t_d \in D'$ *exists such that* $t_d = \lambda_{D'}(Sub(t_q))$ *and* $t_q \preceq_{\mathcal{R}_{\trianglerighteq}} t_d$ *(according to Definition 5.12).* ✦

With Definition 5.15 the simulation can be extended to conjunction, disjunction and negation of ground query terms.

Definition 5.16 (Simulation of Conjuction, Disjunction and Negation of Ground Query Terms in a Set of Normalized Data Terms)**.** *Let $D' \subseteq D$ be a set of normalized data terms, and let t_q and t'_q be ground query terms. Then following holds.*

- $and\{t_q, t'_q\} \preceq_{\mathcal{R}_\trianglerighteq} D'$ *if* $t_q \preceq_{\mathcal{R}_\trianglerighteq} D'$ *and* $t'_q \preceq_{\mathcal{R}_\trianglerighteq} D'$.
- $or\{t_q, t'_q\} \preceq_{\mathcal{R}_\trianglerighteq} D'$ *if* $t_q \preceq_{\mathcal{R}_\trianglerighteq} D'$ *or* $t'_q \preceq_{\mathcal{R}_\trianglerighteq} D'$.
- $not\ t_q \preceq_{\mathcal{R}_\trianglerighteq} D'$ *if for all* $t_d \in D'$ *it is not the case that* $t_q \preceq_{\mathcal{R}_\trianglerighteq} t_d$. ✦

5.4.2 Data Term Selector Revised

During the simulation of a ground query term which has ground query terms at object positions in a dataset G, λ_G selects appropriate data terms by which the simulation can proceed. This includes the search for the first data term in G to start the simulation (cf. Definition 5.6). Thus, a data term selector supersedes root nodes. Whereas root nodes could be easily added to Xcerpt$^{\text{RDF}}$, the simulation does not become easier. Moreover, root nodes are not part of the data model of RDF.

The following Example 5.3 shows how a query is evaluated on cyclic data by simulation using data term selectors.

Example 5.3 (λ_G on a Cyclic Data Term)**.** *Let the cyclic data term $t_d = a\{\, b \rightarrow c\{\, d \rightarrow e\{\, f \rightarrow a\,\}\,\}\,\}$ be given. Furthermore, let the ground query term $t_q = e\{\{\, f \rightarrow a\{\{\, b \rightarrow c\,\}\}\,\}\}$ be given. Let $\mathcal{R}_\trianglerighteq = \{(e,e),(f,f)\}$. First, t_d is normalized to $t_d^1 = a\{\, b \rightarrow c\,\}$, $t_d^2 = c\{\, d \rightarrow e\,\}$ and $t_d^3 = e\{\, f \rightarrow a\,\}$.*

- *$Sub(t_q) = e$ and $\lambda_G(Sub(t_q)) = t_d^3 = e\{\, f \rightarrow a\,\}$. Thus, the simulation starts by showing that $t_q \preceq_{\mathcal{R}_\trianglerighteq} t_d^3$.*
- *$(Sub(t_q), Sub(t_d^3)) \in \mathcal{R}_\trianglerighteq$, so that it remains to show that $f \rightarrow a\{\{\, b \rightarrow c\,\}\} \preceq^{\text{pt}}_{\mathcal{R}_\trianglerighteq} f \rightarrow a$.*
- *$(f,f) \in \mathcal{R}_\trianglerighteq$ and $a\{\{\, b \rightarrow c\,\}\} \in Q_g$, so that it remains to show that $a\{\{\, b \rightarrow c\,\}\} \preceq_{\mathcal{R}_\trianglerighteq} \lambda_G(a)$. Since $\lambda_G(a) = t_d^1 = a\{\, b \rightarrow c\,\}$ it is clear that both terms simulate.*

Altogether, it holds that $t_q \preceq_{\mathcal{R}_\trianglerighteq} t_d$. ✦

According to Example 5.3 the query term is completely read while pseudo-term simulation and data term selection λ_G follows the reading of the query term. If the pseudo-term simulation or λ_G cannot follow, e.g. if the pseudo-terms do not simulate, the simulation fails. The simulation terminates as soon as the query term is completely read because every qery term is finite and probably cyclic query terms have always an acyclic (linear) representation in Xcerpt$^{\text{RDF}}$.

5.4.3 Matching of Blank Nodes Against Blank Nodes by Simulation of their Concise Bounded Descriptions

As the simulation of ground query terms in data terms was intrdoced, the matching between blank nodes can be discussed from another perspective considering concise bounded descriptions (cf. [24]). The matching of a blank node b_1 with another blank node b_2 shall be equivalent to the notion of finding a suitable blank node b_2 for b_1 in the data. For b_1 the blank node b_2 is

suitable when the concise bounded description of b_1 can be found in the concise bounded description of b_2. This could be described more formally in the following proposition that specifies the the matching of blank nodes against blank nodes more specifically as given in Definition 5.5.

Definition 5.17 (Matching of Blank Nodes Against Blank Nodes by Simulation)**.** *A blank node b_1 matches against another blank node b_2 if and only if $cbd(b_1) \preceq_{\mathcal{R}_\trianglerighteq} cbd(b_2)$, where $cbd(b)$ is the concise bounded description of b.* ✦

This definition actually captures the notion between matching between blank nodes in contrast to Definition 5.5 in which it is said that two blank nodes always match. However, the replacement of Definition 5.5 by Definition 5.17 in the simulation as given in Definition 5.12 yield the same simulation results.

In order to show that both definitions of matching between blank nodes do not affect the results of simulation (and hence subsumption), consider two Xcerpt$^{\text{RDF}}$ terms t_1 and t_2, both having blank nodes in subject position, i.e. $Sub(t_1) = b_1 \in B$ and $Sub(t_2) = b_2 \in B$. According to Definition 5.5 b_1 always matches with b_2. Thus, whenever blank nodes are matched according to Definition 5.5, all possible simulation candidates remain to be tried. That is, the matching between blank nodes according to Definition 5.5 has no effect on the simulation results. According to Definition 5.17 the actual pseudo-term simulation of t_1 and t_2 is only done when $cbd(b_1)$ simulates in $cbd(b_2)$. However, since both concise bounded descriptions are subgraphs of or equal to the graphs described by t_1 and t_2, it holds that whenever $t_1 \preceq_{\mathcal{R}_\trianglerighteq} t_2$ it is also the case that $cbd(b_1) \preceq_{\mathcal{R}_\trianglerighteq} cbd(b_2)$.

Definition 5.17 refines the matching between blank nodes by simulating the concise bounded descriptions of t_1 in that of t_2, which, however, is already contained in the simulation of the entire graphs described by t_1 and t_2. Despite this efficiency consideration, one can argue that Definition 5.17 is more elegant than Definition 5.5 in the sense that it captures the notion of matching between blank nodes more directly and profoundly than it is done in Definition 5.5.

5.4.4 About the Shorthand Notations for Containers in Query and Data Terms

As mentioned in Section 4, shorthand notations for RDF containers in data terms are replaced by their respective longhand notations, whereas within query terms these shorthand notations are retained and hence immediately evaluated. The evaluation of container shorthand notations in query terms against their respective longhand notations in data terms, however, leads to complications which are discussed hereafter.

One interesting query example is the querying of a single RDF container in a data term by several Xcerpt$^{\text{RDF}}$ query container pseudo-terms. Consider the following data term which models a tutorial group.

```
eg:tutorialGroup {
  rdf:type  → rdf:Bag
  rdf:_1    → eg:john
  rdf:_2    → eg:jim
  rdf:_3    → eg:tim
  rdf:_4    → eg:anna
  rdf:_5    → eg:julia
}
```

The following query term retrieves all possible partitions of the container into two containers, one consisting of two members and the other consisting of three members.

```
var X {{
  bagOf{{var A, var B}}
  bagOf{{var C, var D, var E}}
}}
```

However, such a query cannot be evaluated because according to Definition 5.12 the simulation of the query term in the data term fails. Due to the injectivity of the simulation, only one bagOf-pseudo-term of the query term can simulate in the data pseudo-term rdf:type → rdf:Bag. An implicit revocation of the injectivity of the mapping for bagOf-pseudo-term to the rdf:type → rdf:Bag would be a remedy. However, this leads to an unintuitive and unsystematic semantics of Xcerpt^{RDF} queries and provides the risk that the user might be hindered in understanding the query evaluation of Xcerpt^{RDF}. Therefore, another approach could be to introduce the revocation of injectivity by delimiters as discussed in Section 3.8 with which the user can decide when bagOf-pseudo-terms shall be mapped without requiring injectivity. Following this approach, by the use of semicolons for this purpose, the query term could then be rewritten as follows.

```
var X {{
  bagOf{{var A, var B}};
  bagOf{{var C, var D, var E}}
}}
```

The semicolon could indicate that both bagOf-pseudo-terms shall simulate in the same pseudo-term rdf:type → rdf:Bag. The element lists of the bagOf-containers would still have to simulate in distinct data pseudo-terms. This approach, however, is ambiguous. Beside the meaning of the semicolon as previously described, the semicolon could also express that the elements of each bagOf-pseudo-term shall be mapped injectively, whereas both element lists may be mapped to the same data pseudo-terms. The latter reading of the semicolon is even likely to be more intuitive than the former reading, since the semicolon would comprise the complete pseudo-terms including their element lists.

This example shows that evaluation of Xcerpt^{RDF} container shorthand notations against their respective RDF longhand notations can be very involved. As a remedy, the shorthand notations of RDF containers could also be retained within data terms. Additionally, it would be necessary to reduce RDF container descriptions to the respective Xcerpt^{RDF} shorthand notations resembling the treatement of container descriptions in Xcerpt^{RDF} query terms. However, in this thesis

such a reduction is not considered. This decision stems from the fact that the shorthand notations of RDF containers exceeds the expressiveness of pure RDF, whereas it has been decided for this thesis that Xcerpt^{RDF} data terms are considered to retain the expressivity of pure RDF.

5.5 Deciding Subsumption - Simulation of Query Terms in Query Terms

This section discusses the simulation of Xcerpt^{RDF} query terms into Xcerpt^{RDF} query terms. This kind of simulation is intended to serve as a method for deciding subsumption. Subsumption imposes an partial ordering of query terms with respect to their generality: A query term q is more general than (or subsumes) another query term q' if and only if all valid answers of q' are also valid for q where validity of answers is defined by the simulation of ground query terms in data terms (cf. Section 5.2). Using the simulation of query terms in data terms, subsumption can be defined as given in Definition 5.18.

Definition 5.18. *A query term q subsumes a query term q' which is written $q \preceq_S q'$ if and only if for all data terms $d \in D$ it holds that whenever $q' \preceq d$ then $q \preceq d$. In other words: let $answer(q) =_{\text{def}} \{d \in D \mid q \preceq d\}$. Then the subsumption is defined as $q \preceq_S q' \Leftrightarrow answer(q') \subseteq answer(q)$* ✦

The latter reading of Definition 5.18 shows that subsumption actually defines a partial ordering of query terms since these properties are inherited from the subset-relation.

Subsumption can be used as an optimization method for query evaluation. Knowing that a query term q subsumes q' avoids a redundant determination of answers for q' since these are already contained in the evaluation of q.

The simulation of query pseudo-terms in query pseudo-terms differs from the pseudo-term simulation of query pseudo-terms in data terms introduced in Definition 5.21. Whereas in Definition 5.12 ground query terms are considered, the simulation between query terms is defined on non-ground query terms (non-ground in the sense that it contains Xcerpt^{RDF} variables). As a consequence, the matching of labels must be extended in order to consider Xcerpt^{RDF} variables.

Definition 5.19 (Matching of Labels Including Xcerpt^{RDF} Variables)**.** *A label l_1 matches against a label l_2, written as $l_1 \trianglerighteq l_2$, if one of the cases as given in Definition 5.5 or one of the following cases hold.*

- *Both l_1 and l_2 are Xcerpt^{RDF} variables, i.e. $l_1, l_2 \in V$.*
- *$l_1 \in V$ and $l_2 \in N$.* ✦

One the one hand, the simulation of Xcerpt^{RDF} container shorthand notations in property-object pseudo-terms is replaced by the simulation of container shorthand notations in container shorthand notations since for simplicity it has been decided to retain existing shorthand notations and to reduce longhand notations to their respective shorthand notations in query terms. On the other hand, additional cases must be considered, e.g. the simulation of negative pseudo-terms in negative pseudo-terms. As another example, in query terms not only labels may occur

as objects but also query terms. The simulation of query terms into query terms given in Defintion 5.20 uses the simulation of query pseudo-terms into query pseudo-terms (cf. Definition 5.21).

Definition 5.20 (Simulation of Query Terms into Query Terms)**.** *Let two query terms q and q' be given. One says that q simulates (via $\mathcal{R}_{\trianglerighteq}$) in q', written as $q \preceq_{\mathcal{R}_{\trianglerighteq}} q'$ if $(Sub(q), Sub(q')) \in \mathcal{R}_{\trianglerighteq}$ and the following holds.*

1. *Case: Both q and q' do not contain negative pseudo-terms $PT^-(q) = \emptyset$ and $PT^-(q') = \emptyset$*
   (a) *A total and injective mapping $\pi\colon PT^+(q) \rightarrow PT^+(q')$ exists such that for all $s \in PT^+(q)$ it holds that $s \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \pi(s)$.*
2. *Case: q contains negative pseudo-terms, i.e. $PT^-(q) \neq \emptyset$, $PT^-(q') = \emptyset$*
   (a) *An total and injective mapping $\pi\colon PT^+(q) \rightarrow PT^+(q')$ as described above exists.*
   (b) *π is not extensible to a $\overline{\pi}\colon (PT^+(q) \cup PT^-(q)) \rightarrow PT^+(q')$ such that for some $s^- \in PT^-(q)$ it holds that $s^- \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \overline{\pi}(s^-)$.*
3. *Case: Both q and q' contain negative pseudo-terms, i.e. $PT^-(q) \neq \emptyset$ and $PT^-(q') \neq \emptyset$*
   (a) *An total and injective mapping $\pi\colon (PT^+(q) \cup PT^-(q)) \rightarrow (PT^+(q') \cup PT^-(q'))$ exists.*
   (b) *On $PT^+(q)$, γ is given as described above.*
   (c) *π is not extensible to a $\overline{\pi}\colon (PT^+(q) \cup PT^-(q')) \rightarrow PT^+(q')$ such that for some $s^- \in PT^-(q)$ it holds that $s^- \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \overline{\pi}(s^-)$ and $\overline{\pi}(s^-) \in PT^+(q') \setminus \text{Image}(\pi)$.*
   (d) *For all $s^- \in PT^-(q)$ it holds that $\pi(s^-) \in PT^-(q')$ and $\pi(s^-) \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} s^-$.* ✦

Beside the syntactic specifities of query terms, simulation between query terms (and thus between query pseudo-terms) is carefully designed to serve as a decision procedure for subsumption between query terms.

Note that in Definition 5.20 optional pseudo-terms are disregarded. From the viewpoint of subsumption this is eligible since optional pseudo-terms never shrink the set of possible answers for a query term. Thus, optional pseudo-terms do not affect subsumption.

The simulation of query pseudo-terms is given in Definition 5.21 and uses the simulation of container elements into container elements as given in Definition 5.22.

Definition 5.21 (Simulation of a Query Pseudo-Term in a Query Pseudo-Term)**.** *Let two query pseudo-terms s_q and s'_q of arbitrary Xcerpt^{RDF} query terms q and q' be given. One says that $s_q \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} s'_q$ if the following holds.*

- *If $s_q \in PT_{\text{PO}}(q)$ then $s'_q \in PT_{\text{PO}}(q')$ and the following statements are valid.*
  1. *$(Prop(s_q), Prop(s'_q)) \in \mathcal{R}_{\trianglerighteq}$*
  2. *If $Obj(s_q) \in N$ then one of the following alternatives hold.*
     (a) *$Obj(s'_q) \in N$ and $(Obj(s_q), Obj(s'_q)) \in \mathcal{R}_{\trianglerighteq}$.*
     (b) *$Obj(s'_q) \in Q_g$ and $(Obj(s_q), Sub(Obj(s'_q))) \in \mathcal{R}_{\trianglerighteq}$.*

3. *If* $Obj(s_q) \in Q_g$*, then* $Obj(s'_q) \in Q_q$ *and* $Obj(s_q) \preceq_{\mathcal{R}_\trianglerighteq} Obj(s'_q)$.

- *If* $s_q \in PT_{\text{cont}}(q)$ *then* $s'_q \in PT_{\text{cont}}(q')$*. Furthermore,* $(\mathcal{U}(s_q), \mathcal{U}(s'_q)) \in \mathcal{R}_\trianglerighteq$ *and the following holds.*
  - *Case:* $\mathcal{E}^-(s_q) = \emptyset$ *and* $\mathcal{E}^-(s'_q) = \emptyset$.
    1. *A total mapping* $\gamma\colon \mathcal{E}^+(s_q) \to \mathcal{E}^+(s'_q)$ *suitable for* s_q *exists.*
    2. *For all* $e \in \mathcal{E}^+(s_q)$ *it holds that* $e \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} \gamma(e)$.
  - *Case:* $\mathcal{E}^-(s_q) \neq \emptyset$ *and* $\mathcal{E}^-(s'_q) = \emptyset$.
    1. *A total mapping* $\gamma\colon \mathcal{E}^+(s_q) \to \mathcal{E}^+(s'_q)$ *as described above exists*
    2. γ *cannot be extended to a* $\overline{\gamma}\colon (\mathcal{E}^+(s_q) \cup \mathcal{E}^-(s_q)) \to \mathcal{E}^+(s'_q)$ *such that for some* $e^- \in \mathcal{E}^-(s_q)$ *it holds that* $e^- \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} \overline{\gamma}(e^-)$.
  - *Case:* $\mathcal{E}^-(s_q) \neq \emptyset$ *and* $\mathcal{E}^-(s'_q) \neq \emptyset$.
    1. *A total mapping* $\gamma\colon (\mathcal{E}^+(s_q) \cup \mathcal{E}^-(s_q)) \to (\mathcal{E}^+(s'_q) \cup \mathcal{E}^-(s'_q))$ *suitable for* s_q *exists*
    2. *On* $\mathcal{E}^+(s_q)$*,* γ *is as described above.*
    3. γ *is not extensible to a* $\overline{\gamma}\colon (\mathcal{E}^+(s_q) \cup \mathcal{E}^-(s_q)) \to \mathcal{E}^+(s'_q)$ *such that* $\overline{\gamma}(e^-) \in \mathcal{E}^+(s'_q) \setminus \text{Image}(\gamma)$ *for some* $e^- \in \mathcal{E}^-(s_q)$.
    4. *For all* $e^- \in \mathcal{E}^-(s_q)$ *it holds that* $\gamma(e^-) \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} e^-$. ✦

Since container elements of container descriptions in query terms are not further simulated in property-object pseudo-terms (cf. Definition 5.14), a new kind of pseudo-term simulation, viz. simulation of container elements in container elements, is necessary which is given in Definition 5.22.

Definition 5.22 (Simulation of Container Elements in Container Elements)**.** *Given two container elements* e_1 *of the form* $pos\ N\ e'_1$ *or* e' *and* e_2 *of the form* $pos\ N\ e'_2$ *or* e'_2*.* e_1 *is said to simulate (via* $\mathcal{R}_\trianglerighteq$*) in* e_2 *which is written* $e_1 \preceq^{\text{elem}}_{\mathcal{R}_\trianglerighteq} e_2$ *if one of the following two cases hold.*

1. *If* $e'_1 \in N \cup V$ *and* $e'_2 \in N \cup V$ *then* $(e'_1, e'_2) \in \mathcal{R}_\trianglerighteq$.
2. *If* $e'_1 \in N \cup V$ *and* $e'_2 \in Q_q$ *then* $(e'_1, Sub(e'_2)) \in \mathcal{R}_\trianglerighteq$.
3. *If* $e'_1 \in Q_g$ *then* $e'_2 \in Q_g$ *and* $e'_1 \preceq_{\mathcal{R}_\trianglerighteq} e'_2$. ✦

The following Example 5.4 shows how simulation of query terms into query terms is computed.

Example 5.4 (Simulation of Query Terms into Query Terms to Decide Subsumption)**.** *Let the query term* q *be given as follows.*

```
1 var U {{
2   var V → var W
3   bagOf{{var X, var Y}}
4   (without eg:knows → var W{{ foaf:knows → eg:jim }})
5 }}
```

Furthermore, let the query term q' be given as follows.

```
1 var A {{
2   foaf:knows → var B
3   bagOf{var C, var D}
4   (without eg:knows → var B)
5 }}
```

We want to show that q simulates into q'.

- *Let $\mathcal{R}_{\trianglerighteq} = \{$(var U, var A), (var V, foaf:knows), (var W, var B), (rdf:Bag, rdf:Bag), (var X, var C), (var Y, var D), (eg:knows, eg:knows), (var B, var W)$\}$. Thus, $\mathcal{R}_{\trianglerighteq}$ satisfies the many-to-one property as requested.*
- *The subjects match, i.e. (var U,var A) $\in \mathcal{R}_{\trianglerighteq}$.*
- *Case 3 of Definition 5.20 holds. Let π be the following mapping.*
  - *var V → var W $\mapsto$ foaf:knows → var B.*
  - *bagOf{{var X, var Y}} $\mapsto$ bagOf{var C, var D}.*
- *var V → var W $\preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}}$ foaf:knows → var B because (var V,foaf:knows) $\in \mathcal{R}_{\trianglerighteq}$ and (var W,var B) $\in \mathcal{R}_{\trianglerighteq}$.*
- *bagOf{{var X, var Y}} $\preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}}$ bagOf{var C, var D} because $\mathcal{U}$(bagOf{{var X, var Y}}) = rdf:Bag, $\mathcal{U}$(bagOf{var C, var D}) =rdf:Bag and (rdf:Bag,rdf:Bag) $\in \mathcal{R}_{\trianglerighteq}$.*
- *Let γ be the suitable mapping for bagOf{{var X, var Y}}, i.e. injective and total as follows.*
  - *var X $\mapsto$ var C and var X $\preceq^{\text{elem}}_{\mathcal{R}_{\trianglerighteq}}$ var C because (var X,var C) $\in \mathcal{R}_{\trianglerighteq}$.*
  - *var Y $\mapsto$ var D and var Y $\preceq^{\text{elem}}_{\mathcal{R}_{\trianglerighteq}}$ var D because (var Y,var D) $\in \mathcal{R}_{\trianglerighteq}$.*
- *(without eg:knows → var W{{ foaf:knows → eg:jim }}) cannot be mapped to any positive pseudo-term in q' because every positive pseudo-term is already in the image of π.*
- *(without eg:knows → var W{{ foaf:knows → eg:jim }}) $\mapsto$ (without eg:knows → var B) and (without eg:knows → var B) $\preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}}$ (without eg:knows → var W{{ foaf:knows → eg:jim }}) because (eg:knows,eg:knows) $\in \mathcal{R}_{\trianglerighteq}$ and (var B,var W) $\in \mathcal{R}_{\trianglerighteq}$.*
- *Thus, π is injective and total as requested.*

Thus, $q \preceq_{\mathcal{R}_{\trianglerighteq}} q'$ and hence q subsumes q'. ✦

Besides the syntactic differences of Xcerpt$^{\text{XML}}$ and Xcerpt$^{\text{RDF}}$, the simulation relations of query terms into query terms of Xcerpt$^{\text{RDF}}$ and Xcerpt$^{\text{XML}}$ also differ in the property 3c in Definition 5.20. This property stipulates that a query term q_1 simulates into a query term q_2 not only if all negative pseudo-terms in q_1 can be mapped injectively to appropriate negative pseudo-terms in q_2 but also if none of the negative pseudo-terms in q_1 can be mapped injectively to a positive pseudo-term unused by the pseudo-term mapping π in q_2, such that they simulate in the sense of 5.21. This is necessary because otherwise, there were query terms which would simulate according to Definition 5.20 but are not in the subsumption hierarchy. Consider the following ground Xcerpt$^{\text{RDF}}$ query terms.

q1

```
a {{
  (without b → c )
  (without d → e )
}}
```

q2

```
a {{
  b → c
  (without b → c )
  (without d → e )
}}
```

Assume that property 3c in Definition 5.20 is not required. Then q_1 simulates into q_2. As the simulation of ground query terms into ground query terms is designed to serve as a decision procedure for subsumption, one expects that q_1 subsumes q_2, i.e. $answer(q_2) \subseteq answer(q_1)$ (cf. Definition 5.18). That is, if q_2 simulates into a data term d then q_1 simulates into d as well. Now, let d be a{b → c}. The query term q_2 simulates into d but q_1 does not. That is, q_1 does not subsume q_2 and thus, the simulation without the property 3c in Definition 5.20 yields an incorrect result.

5.6 Deciding Leanness of RDF Graphs - Simulation of Data in Data Terms

A permissive usage of blank nodes in RDF graphs carries the risk of redundancy. For the formal foundations of RDF, as a kind of graph normalization, as well as for an efficient answering of queries it is considerable to detect these redundancies. Moreover, merging of two RDF graphs can lead to redundant knowledge. For example, consider an RDF graph containing _:x { foaf:name → "John"} and another containing eg:john{foaf:name → "John", foaf:phone → "555−1234"}. The merge of both RDF graphs preserves both data terms, although the knowledge represented by the first data term is already contained in the second data term. Therefore, the notion of lean graphs is introduced (cf. [15]). In this section, it is discussed how simulation of data terms into data terms can be used to check leanness of RDF graphs which are given as sets of Xcerpt^{RDF} data terms. Therefore, Definition 5.23 recapitulates the notion of an RDF graph according to [15].

Definition 5.23 (RDF Graph)**.** *An RDF graph is defined as a set of RDF triples. RDF triples are 3-tuples consisting of a subject, a predicate and an object (also cf. Section 2.1).* ✦

Furthermore, Definition 5.24 recapitulates the notion of lean graphs according to [15].

Definition 5.24 (Lean Graphs (According to [15]))**.** *Let G be an RDF graph. G is called lean if it has no instance G' which is a proper subset of G.*
G' is called an instance of G if there exists a mapping $m\colon B \to N$ such that some or all blank nodes b in G are replaced by $m(b)$. Recall that B denotes the set of blank nodes and N denotes the set of RDF nodes (cf. Definition 5.1). ✦

Intuitively, no subgraph of a lean graph can be omitted without changing the knowledge represented by the graph. The following Example 5.5 illustrates how Definition 5.24 can be used to show the non-leanness of an RDF graph.

Example 5.5 (Leanness of RDF Graphs)**.** *Let $G_1 = \{$(eg:a, eg:b, eg:c), (_:X, eg:b, eg:c), (_:Y, eg:b, eg:c)$\}$ be an RDF graph. To show that G_1 is not lean we have to find a mapping $m\colon B \to N$ which yields an instance $G' \subset G_1$. Let m be as follows: _:X $\mapsto$ eg:a, _:Y $\mapsto$ eg:a. Hence we get*

$G' = \{(eg{:}a,\ eg{:}b,\ eg{:}c)\}$ and $G' \subset G_1$. Thus, G_1 is not lean.
As another example, let $G_2 = \{(\_{:}X,\ p,\ \_{:}Y), (\_{:}Y,\ p,\ \_{:}X)\}$. This graph is lean, since there's no mapping which yields an instance being a proper subset of G_2:

- *$\_{:}X \mapsto \_{:}Y$ yields the triple $(\_{:}Y,\ p,\ \_{:}Y) \notin G_2$*
- *$\_{:}Y \mapsto \_{:}X$ yields the triple $(\_{:}X,\ p,\ \_{:}X) \notin G_2$*
- *The combination of both also does not yield an instance being a proper subset of G_2.*

Thus, G_2 is lean. ✦

Transferred to Xcerpt$^{\text{RDF}}$, leanness concerns sets of normalized data terms. For example, the data set

```
1  _:someone { foaf:knows → _:someoneElse},
2  eg:john { foaf:knows → eg:anna }
```

is not lean, since the latter data term already contains the knowledge expressed by the first data term. The latter data term entails the first data term. Thus, the first data term can be omitted.

In this section, a special kind of simulation-relation is introduced to determine (non-)leanness of sets of normalized Xcerpt$^{\text{RDF}}$ data terms. But therefore, it is necessary to refer to those data terms which have as object labels the subject of a given data term d. Consider the following Definition 5.25.

Definition 5.25. *Let an arbitrary set of normalized data terms $D' \subseteq D$ be given. Let $d \in D'$. Define $\delta(d)$ as the set of normalized data terms in D' in which the subject of d occurs as object labels: $\delta(d) =_{\text{def}} \{d' \in D' | \exists s \in PT(d') \colon Obj(s) = Sub(d)\}$* ✦

With the Definition 5.25 the simulation of data terms into data terms can be defined as given in Definition 5.26.

Definition 5.26 (Simulation of Data Terms into Data Terms)**.** *Let an arbitrary set of normalized data terms $D' \subseteq D$ and two normalized data terms $d_1, d_2 \in D'$ be given. d_1 simulates (via $\mathcal{R}_{\trianglerighteq}$) in d_2, written as $d_1 \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d_2$, if the following holds.*

1. *$(Sub(d_1), Sub(d_2)) \in \mathcal{R}_{\trianglerighteq}$*
2. *A total mapping $\pi \colon PT(d_1) \to PT(d_2)$ exists such that for all $s \in PT(d_1)$ it holds that $s \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \pi(s)$. If $d_1 = d_2$ then π is not the identity mapping.*
3. *If there is an object label o_i in d_1 which is subject of a data term $d \in D'$, then for o_j in d_2 with $(o_i, o_j) \in \mathcal{R}_{\trianglerighteq}$ there also exists a data term $d' \in D'$ with subject o_j and $d \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d'$.*
4. *The domain and the range of $\mathcal{R}_{\trianglerighteq}$ without identity mappings are disjunct . That is, with $\mathcal{R}^{\neq}_{\trianglerighteq} =_{\text{def}} \mathcal{R}_{\trianglerighteq} \setminus \{(l,l) \,|\, (l,l) \in \mathcal{R}_{\trianglerighteq}\}$ it holds that $\text{domain}(\mathcal{R}^{\neq}_{\trianglerighteq}) \cap \text{range}(\mathcal{R}^{\neq}_{\trianglerighteq}) = \emptyset$.*
5. *For all $d_1, d_2 \in D'$ it holds that if $d_1 \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d_2$ then for each $d \in \delta(d_1)$ a data term $d' \in \delta(d_2)$ exists such that $d \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d'$.* ✦

The properties 1 and 2 are similar to the other kinds of simulation-relations as described in the sections above. However, the restriction in property 2 that π may not be the identity mapping when a data term is simulated into itself is necessary since otherwise data terms always simulate into themselves. Concerning the property 3 consider the following Example 5.6.

Example 5.6 (Necessity of the Property 3 in Definition 5.26)**.** *Consider the RDF graph* $G = \{(\textit{eg:a, eg:b, eg:c }), (\textit{eg:a, eg:b, \_:X}), (\textit{\_:X, eg:b, eg:d})\}$*. According to Definition 5.24 the RDF graph* G *is lean. Rewrite* G *as the following set of normalized data terms* D'*.*

$$
\begin{array}{lll}
1 & \textit{eg:a} \ \{ \ \textit{eg:b} \rightarrow \textit{eg:c}\,, & (d_1) \\
2 & \qquad\quad \textit{eg:b} \rightarrow \textit{\_:X} \ \} & \\
3 & \textit{\_:X} \ \{ \ \textit{eg:b} \rightarrow \textit{eg:d} \ \} & (d_2)
\end{array}
$$

Without the property 3 in Definition 5.26 the data term d_1 *simulates into itself because*

- *Let* $\mathcal{R}_{\trianglerighteq} = \{(\textit{eg:a, eg:a}), (\textit{eg:b, eg:b}), (\textit{\_:X, eg:c}), (\textit{eg:c, eg:c})\}$
- $(\textit{eg:a, eg:a}) \in \mathcal{R}_{\trianglerighteq}$
- $\textit{eg:b} \rightarrow \textit{\_:X} \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \textit{eg:b} \rightarrow \textit{eg:c}$
- $\textit{eg:b} \rightarrow \textit{eg:c} \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \textit{eg:b} \rightarrow \textit{eg:c}$
- *The pseudo-term mapping is not the identity mapping.*

However, it is disregarded that for $\textit{\_:X}$ *additional knowledge is given by the data term* d_2 *whereas for* $\textit{eg:c}$ *nothing further is known. Thus, it cannot be assumed that* $\textit{\_:X}$ *and* $\textit{eg:c}$ *denote the same resource. With the property 3 in Definition 5.26 the existence of additional knowledge about* $\textit{\_:X}$ *is taken into consideration and, hence, the set of data terms is correctly determined as lean.* ✦

Concerning the property 4 the following Example 5.7 serves as a justification.

Example 5.7 (The Necessity of the Property 4 in Definition 5.26)**.** *Consider the RDF Graph* $G = \{(\textit{\_:X, eg:p, \_:Y}), (\textit{\_:Y, eg:p, \_:X})\}$*. In Example 5.5 it has been shown, that* G *is lean. Rewrite* G *as the following set of normalized data terms* D'*:*

$$
\begin{array}{lll}
1 & \textit{\_:X} \ \{ \ \textit{eg:p} \rightarrow \textit{\_:Y} \ \} & (d_1) \\
2 & \textit{\_:Y} \ \{ \ \textit{eg:p} \rightarrow \textit{\_:X} \ \} & (d_2)
\end{array}
$$

We show that d_1 *simulates into* d_2 *(by now, disregarding the non-termination which is discussed later):*

- *Let* $\mathcal{R}_{\trianglerighteq} = \{(\textit{\_:X, \_:Y}), (\textit{eg:p, eg:p}), (\textit{\_:Y, \_:X})\}$
- $(\textit{\_:X, \_:Y}) \in \mathcal{R}_{\trianglerighteq}$
- $\textit{eg:p} \rightarrow \textit{\_:Y} \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \textit{eg:p} \rightarrow \textit{\_:X}$*, since* $(\textit{eg:p, eg:p}) \in \mathcal{R}_{\trianglerighteq}$ *and* $(\textit{\_:Y, \_:X}) \in \mathcal{R}_{\trianglerighteq}$*.*
- *For* $\textit{\_:Y}$ *the data term* d_2 *and for* $\textit{\_:X}$ *the data term* d_1 *exists and* $d_2 \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d_1$*.*

Thus, without property 4 the data set is supposed to be not lean. Considering the resulting relation $\mathcal{R}_{\trianglerighteq}$, removing identity mappings, $\mathcal{R}_{\trianglerighteq}^{\neq} = \{(\_{:}X,\ \_{:}Y), (\ \_{:}Y,\ \_{:}X\ )\}$ whose domain and range are not disjunct. Thus, to guarantee that simulation yields the correct result, property 4 must be requested in Definition 5.26. ✦

The property 4 excludes identity mappings since these unavoidably appear when simulating data terms into themselves, which is eligible.

However, not all data terms which simulate into others (by the properties 1 to 4 in Definition 5.26) make sets of normalized data terms non-lean. The reason for this is that for a data term d references by other data terms are disregarded. From the RDF graph perspective, incoming edges are disregarded. This issue is discussed in Example 5.8.

Example 5.8 (Necessity of Property 5 in Definition 5.26)**.** *Let the lean RDF graph $G = \{$ ($\_{:}A$, eg:r, eg:m), ($\_{:}A$, eg:p, $\_{:}C$), ($\_{:}B$, eg:p, $\_{:}C$), (eg:m, eg:q, $\_{:}B$) $\}$ be given. Rewrite the RDF graph G as the following data set D':*

$$
\begin{array}{lllll}
1 & \_{:}A & \{\ eg:r & \rightarrow\ eg:m & (d_1)\\
2 & & eg:p & \rightarrow\ \_{:}C\ \} & \\
3 & \_{:}B & \{\ eg:p & \rightarrow\ \_{:}C\ \} & (d_2)\\
4 & eg:m & \{\ eg:q & \rightarrow\ \_{:}B\ \} & (d_3)
\end{array}
$$

However, this data set shows to be not lean in the sense of Definition 5.26 excluding property 2 because d_2 simulates into d_1 (again, by now, disregarding non-termination):

- *Let $\mathcal{R}_{\trianglerighteq} = \{(\_{:}B,\ \_{:}A), (eg{:}p,\ eg{:}p), (\_{:}C,\ \_{:}C)\}$.*
- *$(\_{:}B,\ \_{:}A) \in \mathcal{R}_{\trianglerighteq}$*
- *$eg{:}p \rightarrow \_{:}C \preceq^{\mathrm{pt}}_{\mathcal{R}_{\trianglerighteq}} eg{:}p \rightarrow \_{:}C$*

The reference of d_3 to d_2 via eg:q which makes the graph lean is disregarded. In the given data set it is necessary that each data term in which d_2 occurs as object must also have a simulation partner. This is formalized by property 5 in Definition 5.26. Accordingly, these data terms are comprised by $\delta(\_{:}B\{\ eg{:}p \rightarrow \_{:}C\ \})$ which evaluates to $\{\ eg{:}m\{eg{:}q \rightarrow \_{:}B\}\ \}$ for which no data term $d' \in D'$ exists such that d' simulates into $eg{:}m\{eg{:}q \rightarrow \_{:}B\}$. Thus, according to Definition 5.26 (including property 2) D' is recognized as lean. ✦

One issue of Definition 5.26 which appeared in the examples above is that the simulation of data terms does not always terminate. Therefore, consider the following set of normalized data terms.

$$
\begin{array}{lllll}
1 & \_{:}\mathrm{x} & \{\ \mathrm{eg{:}b} & \rightarrow\ \_{:}\mathrm{x} & \}\quad (d_1)\\
2 & \mathrm{eg{:}a} & \{\ \mathrm{eg{:}b} & \rightarrow\ \mathrm{eg{:}a} & \}\quad (d_2)
\end{array}
$$

With $\mathcal{R}_{\trianglerighteq} = \{(\_{:}\mathrm{x},\ \mathrm{eg{:}a}), (\mathrm{eg{:}b},\ \mathrm{eg{:}b}\ )\}$ it is $(\_{:}\mathrm{x},\ \mathrm{eg{:}a}) \in \mathcal{R}_{\trianglerighteq}$. Furthermore, $\mathrm{eg{:}b} \rightarrow \_{:}\mathrm{x} \preceq^{\mathrm{pt}}_{\mathcal{R}_{\trianglerighteq}} \mathrm{eg{:}b} \rightarrow \mathrm{eg{:}a}$. For $\_{:}\mathrm{x}$ as well as for $\mathrm{eg{:}a}$ data terms exists, viz. d_1 and d_2 which have to be tested next. That is, in order to show that d_1 simulates into d_2 one has to show that d_1 simulates into d_2.

This example shows that it is rational to consider a cyclic simulation to terminate with result $true$ whenever the cycle only depends on itself. The example above shows the smallest possible cycle. As the simulation of d_1 into d_2 only depends on itself the simulation can be aborted and considered as terminating with result $true$. If the simulation of any member of such cycle additionally depends on a simulation which is not part of that cycle this dependency is decisive for the simulation to succeed or to fail.

The simulation of data terms into data terms is constructed in a way that it can be used to check non-leanness of sets of normalized data terms. This idea is summarized in Definition 5.27.

Definition 5.27 (Non-Leanness of a Set of Normalized Xcerpt$^{\text{RDF}}$ Data Terms)**.** *A set of normalized Xcerpt^{RDF} data terms which is denoted by D' is called non-lean if there are two data terms $d_1, d_2 \in D'$ such that $d_1 \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d_2$.* ✦

The following Example 5.9 shall demonstrate the recognition of non-leanness of graphs (and thus sets of Xcerpt$^{\text{RDF}}$ normalized data terms) according to 5.27.

Example 5.9 (Determining Non-Leanness of an RDF Graph)**.** *Let the RDF graph $G = \{$(eg:a, eg:b, eg:c), (eg:a, eg:b, _:x), (_:x, eg:b, _:y), (eg:c, eg:b, eg:e)$\}$ be given. G is not lean. Let m be as follows: _:x $\mapsto$ c, _:y $\mapsto$ e. The resulting instance of G is $G' = \{$(a,b,c), (c,b,e)$\}$ and $G' \subset G$. Hence, G is not lean.*

Rewrite G as the Xcerpt^{RDF} data set D' as follows.

```
1  eg:a { eg:b → eg:c       }   (d1)
2         eg:b → _:x        },
3  _:x  { eg:b → _:y        },  (d2)
4  eg:c { eg:b → eg:e       }   (d3)
```

d_3 does neither simulate into itself nor does it simulate into d_2. d_2 does not simulate into itself nor into d_1. The latter can be seen as follows: Mapping eg:b → _:y in d_2 to eg:b → eg:c in d_1 requires that for _:y a data term exists which simulates into d_3. This is not the case. The same holds for mapping eg:b → _:y to d_2 to eg:b → _:x in d_1.

But d_1 simulates into itself: $\pi($ eg:b → eg:c $)$ = eg:b → _:x and $\pi($ eg:b → _:x $)$ = eg:b → _:x. Note that according to Definition 5.26 injectivity of π is not required. For _:x and for eg:c data terms exist which simulate as shown above. Furthermore, $\delta(d_1) = \emptyset$, i.e. eg:a does not occur as an object within a data term in D'. Hence, the property 2 in Definition 5.27 is trivially satisfied.

Besides, d_2 simulates into d_3: _:x matches against eg:c and eg:b → _:y $\preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}}$ eg:b → eg:e because eg:b matches against itself and _:y matches against eg:e. Furthermore, $\delta($_:x{eg:b → _:y}$) = \{d_1\}$ which we saw that it simulates into itself. Thus, D' is not lean. ✦

We introduced the simulation of data terms into data terms with several examples illustrating the necessity of the properties specified in Definition 5.26, and we defined non-leanness of sets of normalized Xcerpt$^{\text{RDF}}$ data terms by means of the simulation of data terms into data terms. Finally, it remains to show that non-leanness of RDF graphs (according to Definition 5.24) is equivalent to non-leanness of the corresponding sets of normalized Xcerpt$^{\text{RDF}}$ data terms (according to Definition 5.27). This is stated in the following Proposition 5.1.

Proposition 5.1 (Equivalence of Non-Leanness of RDF Graphs and Non-Leanness of Sets of Normalized Xcerpt$^{\text{RDF}}$ Data Terms)**.** *Let G be an arbitrary RDF graph and let D' be the corresponding set of normalized Xcerpt^{RDF} data terms. G is not lean (in the sense of Definition 5.24) if and only if D is not lean (in the sense of Definition 5.27).*

Proof. "if": Let G be a non-lean RDF graph. Then there exists a mapping $m\colon B \to N$ which creates an instance of G, denoted as G', in replacing all or some blank nodes b in G by $m(b)$, such that $G' \subset G$. There, m cannot be the identity mapping since all triples in G are syntactically different.
Now, translate G to a set of normalized Xcerpt$^{\text{RDF}}$ data terms D': for all triples (s, p_1, o_1), (s, p_2, o_2), $\ldots, (s, p_n, o_n) \in G$ add to D' the data term $s\{p_1 \to o_1, p_2 \to o_2, \ldots, p_n \to o_n\}$. Furthermore, construct the finite matching relation $\mathcal{R}_{\trianglerighteq}$ such that the following is satisfied:

- For all blank nodes b occurring in G for which $m(b)$ is defined, $(b, m(b)) \in \mathcal{R}_{\trianglerighteq}$.
- For all nodes n occurring in G for which $m(n)$ is undefined, $(n, n) \in \mathcal{R}_{\trianglerighteq}$.
- m may contain chains of the form $x \mapsto x_1 \mapsto x_2 \mapsto \ldots \mapsto x_{n-1} \mapsto x_n$ which, however, are equivalent to $x \mapsto x_n$. If m contains such a chain, replace the resulting chain $(x, x_1), (x_1, x_2), \ldots, (x_{n-1}, x_n)$ in $\mathcal{R}_{\trianglerighteq}$ by (x, x_n).

We have to show, that with the constructed $\mathcal{R}_{\trianglerighteq}$, D' is recognized as not lean.
Since G is not lean, there exist triples t_i in G which are made equal to other triples t'_j in G by application of m. Let d_1 be the data term resulting from the translation of these t_i sharing the same subject s and let d_2 be the corresponding data term resulting from the translation of these t'_j sharing the same subject s'. By the choice of the t_i and t_j, either $s = s'$ or $m(s) = s'$. We show that the corresponding Xcerpt$^{\text{RDF}}$ data terms d_1 and d_2 satisfy the properties requested in Definition 5.27. In order to prove that $d_1 \preceq^{\text{L}}_{\mathcal{R}_{\trianglerighteq}} d_2$ we have to show that the requirements given in Definition 5.26 are satisfied:

- *Property 1)* By construction of $\mathcal{R}_{\trianglerighteq}$, $(Sub(d_1), Sub(d_2)) \in \mathcal{R}_{\trianglerighteq}$.
- *Property 2)* A total pseudo-term mapping $\pi\colon PT(d_1) \to PT(d_2)$ exists such that for each $p_1 \in PT(d_1)$ it is $p_1 \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} \pi(p_1) = p_2 \in PT(d_2)$, i.e. $(Prop(p_1), Prop(p_2)) \in \mathcal{R}_{\trianglerighteq}$ and $(Obj(p_1), Obj(p_2)) \in \mathcal{R}_{\trianglerighteq}$. We show this by contradiction. Assume that there exists a pseudo-term $p^* \to o^*$ in d_1 which cannot be mapped to a $p'^* \to o'^*$ in d_2 such that $p^* \to o^* \preceq^{\text{pt}}_{\mathcal{R}_{\trianglerighteq}} p'^* \to o'^*$. For G, this would mean that there exists a triple $t = (s, p^*, o^*)$ which is mapped by m to $t^* = (s', p'^*, o'^*)$ in G' and $t \neq t^*$. However, this contradicts the assumption that m makes the triples corresponding to d_1 equal to the triples corresponding to d_2. Consider the case $d_1 = d_2$. If π is the identity, all triples being translated to d_1 would be mapped to themselves. This would only be possible when $\mathcal{R}_{\trianglerighteq}$ relates labels to themselves, and thus, m would have to map the nodes of these triples to themselves. However, this would mean that G contains duplicates which is impossible, since G is a set.
- *Property 3)* We show this by contradiction. Assume that for an object label o in d_1 there exists a data term $d \in D'$ with $Sub(d) = o$ but for the object label o' in d_2 with $(o, o') \in \mathcal{R}_{\trianglerighteq}$ there does not exist a data term $d' \in D'$ with $Sub(d') = o'$. For the RDF graph G this would mean that m maps the triples $t_i = (o, p_i, n_i) \in G$ to $t'_i = (o', p_i, n'_i) \in G'$ but $t'_i \notin G$, which contradicts the assumption that $G' \subset G$. Now, assume that for o' a data term $d' \in D'$

with $Sub(d') = o'$ exists but $d \not\preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d'$. However, as a consequence there would exist at least one triple t in G which is mapped by m to a triple t' in G' which is not in G. This contradicts the assumption that G is not lean.

- *Property 4)* $\mathcal{R}_\trianglerighteq$ satisfies this property by construction.
- *Property 5)* This can be shown by contradiction. Assume that for some $d \in \delta(d_1)$ there is no $d' \in \delta(d_2)$ such that $d \preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d'$. Then, again, for some triples t the mapping m would yield triples $t' \in G'$ which are not in G, and hence G' would not be a proper subset of G which contradicts the assumption that G is not lean.

"only if": Let D' be a non-lean set of normalized Xcerpt$^{\mathrm{RDF}}$ data terms. That is, there exists a matching relation $\mathcal{R}_\trianglerighteq$ for d_1 and d_2 such that $d_1 \preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d_2$.
Construct an RDF graph G corresponding to D' in a straightforward manner: for each data term $s\{p_1 \rightarrow o_1, p_2 \rightarrow o_2, \ldots, p_n \rightarrow o_n\} \in D'$ add to G the triples $(s, p_1, o_1), (s, p_2, o_2), \ldots, (s, p_n, o_n)$. Construct the mapping m required in Definition 5.24 such that for all $(x, y) \in \mathcal{R}_\trianglerighteq$ with $x \neq y$, $m(x) = y$. We have to show that m creates an instance of G, denoted by G', such that G' is a proper subset of G. This is done by disproving that *1)* $G' = G$ and *2)* $G' \not\subseteq G$. As a consequence it remains that $G' \subset G$.

- *Ad 1)* Assume that $G' = G$, i.e. all triples in G are mapped to themselves. That is, by construction, $m = \emptyset$ because $\mathcal{R}_\trianglerighteq$ is the identity and by construction m does not contain identity mappings. Assume that $m \neq \emptyset$, i.e. there exists a triple $t \in G$ which is mapped by the application of m to a triple t_1. However, to maintain $G' = G$, t_1 has to be mapped to a t_2 which in turn has to be mapped to another t_3. This chain must continue because otherwise there would be two triples which are made equal by the application of m, thus, yielding $G' \neq G$. However, this chain of mappings cannot continue infinitely when we assume that G is finite. That is, there must exist a t_n within the chain which is finally mapped to t. However, the chain $t \mapsto t_1 \mapsto t_2 \mapsto \ldots \mapsto t_n \mapsto t$ is equivalent to $t \mapsto t$. m cannot contain such a chain because it is constructed according to $\mathcal{R}_\trianglerighteq$ and the intersection of the domain and range of $\mathcal{R}_\trianglerighteq$ is disjunct. Hence $m = \emptyset$. Because of $\mathcal{R}_\trianglerighteq$, for all $d \in D'$, $d \preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d$. As $\mathcal{R}_\trianglerighteq$ is the identity, the subject of d matches with itself and the pseudo-term mapping π must also be the identity mapping. However, this violates the property 2 in Definition 5.26.
- *Ad 2)* Assume that $G' \not\subseteq G$. Then there exists a triple t_m which results from the application of m to a triple $t \in G$ but $t_m \notin G$. Let d be the normalized Xcerpt$^{\mathrm{RDF}}$ data term whose translation led to t. As we assume that t_m results from the application of m, which is constructed by $\mathcal{R}_\trianglerighteq$, d must be affected by $\mathcal{R}_\trianglerighteq$. Since $\mathcal{R}_\trianglerighteq$ is constructed during the simulation of d_1 into d_2, d must be considered in this simulation (because of the properties 3 and 5 in Definition 5.26). Since $t_m \notin G$ there cannot be a data term $d' \in D'$ such that $d \preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d'$. However, this violates one of the properties 3 and 5 in Definition 5.26. Hence, we would have to conclude that $d_1 \not\preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d_2$ contradicting the assumption that $d_1 \preceq^{\mathrm{L}}_{\mathcal{R}_\trianglerighteq} d_2$. Thus, G' must be a proper subset of G. ✦

Chapter 6

Conclusion and Pending Issues

The contributions of this thesis can be subdivided into two parts. First, a new syntax for RDF querying and authoring capabilities is introduced. As in Xcerpt^{XML}, data terms serve as an abstract representation of RDF data in Xcerpt^{RDF}. The intuitive syntax of Xcerpt^{RDF} is adapted from TRIPLE [23] because it supports two perspectives valuable for the modelling: the graph based view, where the arrows symbolize the directed edges in an RDF graph and a slot-oriented notation supporting object-oriented design. This basic syntax is systematically enriched in various shorthand notations such as composition, numerous factorizations, and syntactic sugar for RDF specifities like type descriptions for nodes and properties, reification and containers as well as collections. This syntax offers convenience in writing RDF to a great extent such that Xcerpt^{RDF} certainly bears comparison with any other RDF serialization format.
Furthermore, the syntax of query terms introduces all well-known and established query constructs of Xcerpt^{XML} to Xcerpt^{RDF}. This comprises not only the specification of incompleteness, but also optional and negative property-object pairs. There are two types of variables introduced to Xcerpt^{RDF}: label variables which can be bound to RDF nodes and CBD variables which can be bound to concise bounded descriptions of RDF nodes. Moreover, this thesis extensively investigates different kinds of querying RDF containers and shows that the shorthand notations for an RDF container are more expressive than pure RDF. This necessitates to rather evaluate these shorthand notations than their respective longhand notations in query terms.

Second, it is investigated how queries can be evaluated using simulation unification. This is achieved by consideration of a reasonable subset of Xcerpt^{RDF} terms. Data terms are normalized in order to keep the evaluation as simple as possible. The normalization comprises the expansion of shorthand notations and factorizations as well as the lifting of subjects. It is shown that the rules of normalization terminate and are confluent.
The evaluation uses three different ways of simulation: the simulation of ground query terms in data terms to determine answers of queries, the simulation of ground query terms in ground query terms to determine subsumption and, last but not least, the simulation of data terms in data terms to determine leanness of sets of data terms. This thesis also shows that determining the leanness of sets of data terms defined by simulation is equivalent to determining the leanness of the corresponding RDF graph according to [15].
The simulation of ground query terms in data terms is developed with capabilities to evaluate shorthand notations for containers in query terms. The simulation of ground query terms in

ground query terms has shown to be an appropriate and intuitive way of deciding subsumption. The development of this kind of simulation also shows that the simulation of ground query terms in ground query terms as specified in Xcerpt^{XML} must be corrected in order to decide subsumption. Finally, it is shown that the simulation of data terms in data terms as described in this thesis is appropriate to determine leanness of RDF graphs.

Finally, the thesis also comprises a prototypical implementation of the simulation of ground query terms in data terms and the simulation of query terms in query terms. Because of time limitations the simulation of data terms in data terms could not be finished in time. This implementation therefore is meant to serve as a proof of concept.

This thesis is the first step towards querying RDF in Xcerpt. That is why several issues are either not covered by this thesis, e.g. due to time limitations, or are only marginally described because of simplicity. Thus, several issues are pending:

The Evaluation of Shorthand Notations of RDF Collections And the desc Construct The evaluation of RDF collections, i.e. RDF lists, is much more involved than the evaluation of container shorthand notations because the elements of a list are not within a single data term but distributed within the set of data terms. For example, to query whether an element is within a list the evaluation must follow the rdf:rest-edges within the data terms to an arbitrary length. This is infeasible without the notion of incompleteness in depth of Xcerpt^{XML} which is expressed by the language construct desc. The introduction of the (qualified) descendant construct in forthcoming versions of Xcerpt^{RDF} will certainly increase the expressivity of Xcerpt^{RDF} because it enables for example the querying of transitive closures.

The Evaluation of Concise Bounded Descriptions This thesis does not specify the meaning of construct terms which consist only of a CBD variable or in which CBD variables occur at subject or object positions. In conjunction with the depth counter introduced in Section 3.4.1 the evaluation is likely to be very involved. Probably, the evaluation also makes use of a kind of qualified descendant construct in which a maximum depth can be specified.

The Evaluation of Optional and Negative Pseudo-Terms with all Possible Scopes In the evaluation described in this thesis only property-object pseudo-terms of the {without p → o} and {optional p → o} are considered. Due to time limitations the other scopes given in Section 3.4 could not be considered.

Further Element Specifications for Container Shorthand Notations in Query Terms As it is mentioned in Section 3.4.2 the different kinds of brackets introduced by Xcerpt^{XML} do not suffice because there are further eligible queries given in Section 3.4.2 which cannot be expressed in Xcerpt^{RDF} without conflicting with the meanings of the diverse bracketing used in Xcerpt^{XML}.

Unordered Queries for Ordered Container and Collections Due to limitations in time this thesis does not specify how unordered queries can be included to the evaluation of Xcerpt^{RDF} queries.

Acknowledgements.

This research has been funded by the European Commission and by the Swiss Federal Office for Education and Science within the 6th Framework Programme project REWERSE number 506779 (cf. `http://rewerse.net`).

Bibliography

[1] B. Adida, M. Birbeck, *RDFa Primer Embedding Structured Data in Web Pages*, W3C Working Draft, October 2007, http://www.w3.org/TR/xhtml-rdfa-primer/

[2] Ola Andersson et al., *Scalable Vector Graphics (SVG) 1.1 Specification*, W3C Recommendation, January 2003, http://www.w3.org/TR/SVG/

[3] D. Beckett, *Turtle – Terse RDF Triple Language*, November 2007, http://www.dajobe.org/2004/01/turtle/

[4] D.Beckett, B. McBride, *RDF/XML Syntax Specification (Revised)*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-syntax-grammar/

[5] T. Berners-Lee, *Primer: Getting into RDF & Semantic Web using N3*, August 2005, http://www.w3.org/2000/10/swap/Primer

[6] T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifier (URI): Generic Syntax*, RFC 3986, January 2005, http://tools.ietf.org/html/rfc3986

[7] O. Bolzer, *Towards Data-Integration on the Semantic Web: Querying RDF with Xcerpt*, Diploma Thesis, 2005, Institute for Informatics, University of Munich, http://www.pms.ifi.lmu.de/publikationen/diplomarbeiten/Oliver.Bolzer/Oliver.Bolzer.DA.pdf

[8] M. Birbeck, S. Pemperton, B. Adida, *RDF/A Syntax A collection of attributes for layering RDF on XML languages*, W3C Editor's Draft, October 2005, http://www.w3.org/2001/sw/BestPractices/HTML/2005-rdfa-syntax

[9] D. Brickley, B. McBride, *RDF Vocabulary Description Language 1.0: RDF Schema*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-schema/

[10] D.Brickley, L. Miller, *FOAF Vocabulary Specification 0.91*, Namespace Document, November 2007, http://xmlns.org/foaf/0.1/,

[11] J. Broekstra, A. Kampman, *SeRQL: An RDF Query and Transformation Language*, International Semantic Web Conference, ISWC 2004, August 2004

[12] F. Bry, T. Furche, L. Badea, C. Koch, S. Schaffert, S. Berger, *Querying the Web Reconsidered: Design Principles for Versatile Query Languages*, Journal of Semantic Web and Information Systems (IJSWIS) 1 (2), April-June 2005, http://www.pms.ifi.lmu.de/publikationen/PMS-FB/PMS-FB-2005-3.pdf

[13] F. Bry, T. Furche, B. Linse, *Data Model an Query Constructs for Versatile Query Languages: State-of-the-Art and Challenges for Xcerpt*, Proceedings of 4th Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2006), Budva, Montenegro (10th - 11th June 2006), LNCS 4187, 2006, Springer-Verlag, http://www.pms.ifi.lmu.de/publikationen/PMS-FB/PMS-FB-2006-19/PMS-FB-2006-19.pdf

[14] J. Carroll, Ch. Bizer, P. Hayes, P. Stickler, *Named Graphs, Provenance and Trust*, WWW 2005, May 2005, Japan, http://www2005.org/cdrom/docs/p613.pdf

[15] P. Hayes, *RDF Semantics*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-mt/

[16] M. Kifer, G. Lausen, J. Wu, *Logical Foundations of Object-Oriented and Frame-Based Languages*, Journal of the Association for Computing Machinery, May 1995, http://www.cs.umbc.edu/771/papers/flogic.pdf

[17] G. Kline, J. J. Carroll, *Resource Description Framework (RDF): Concepts and Abstract Syntax*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-concepts/

[18] D. König, *Theorie der Endlichen und Unendlichen Graphen: Kombinatorische Topologie der Streckenkomplexe*, Akademischer Verlag, Leipzig, 1936

[19] B. McBride, *RDF Primer*, W3C Recommendation, February 2004, http://www.w3.org/TR/rdf-primer/

[20] A. Philips, M. Davis (Eds.), *Tags for Identifying Languages*, RFC 4646, September 2006, http://www.rfc-editor.org/rfc/rfc4646.txt

[21] Eric Prud'hommeaux, Andy Seaborne, *SPARQL Query Language for RDF*, W3C Recommendation, January 2008, http://www.w3.org/TR/rdf-sparql-query

[22] S. Schaffert, *Xcerpt: A Rule-Based Query and Transformation Language for the Web*, PhD Thesis, 2004, Institute for Informatics, University of Munich, http://www.pms.ifi.lmu.de/publikationen/dissertationen/PMS-DISS-2004-1/ Schaffert_Sebastian.pdf

[23] M. Sintek, S. Decker, *TRIPLE—An RDF Query, Inference, and Transformation Language*, DDLP'2001, Japan, October 2001, http://triple.semanticweb.org/doc/ddlp2001/TripleReport.pdf

[24] P. Stickler, *CBD - Concise Bounded Description*, W3C Member Submission, June 2005, http://www.w3.org/Submission/CBD/

[25] N. Walsh, *Using Qualified Names (QNames) as Identifiers in XML Content*, W3C Tag Finding, March 2004, http://www.w3.org/2001/tag/doc/qnameids.html

[26] http://www.microformats.org

[27] http://infomesh.net/2002/rdfinhtml/