# Functions over RDF Language Elements

Bernhard Schandl

University of Vienna
Department of Distributed and Multimedia Systems
`bernhard.schandl@univie.ac.at`

**Abstract.** RDF data are usually accessed using one of two methods: either, graphs are rendered in forms perceivable by human users (e.g., in tabular or in graphical form), which are difficult to handle for large data sets. Alternatively, query languages like SPARQL provide means to express information needs in structured form; hence they are targeted towards developers and experts. Inspired by the concept of spreadsheet tools, where users can perform relatively complex calculations by splitting formulas and values across multiple cells, we have investigated mechanisms that allow us to access RDF graphs in a more intuitive and manageable, yet formally grounded manner. In this paper, we make three contributions towards this direction. First, we present RDFunctions, an algebra that consists of mappings between sets of RDF language elements (URIs, blank nodes, and literals) under consideration of the triples contained in a background graph. Second, we define a syntax for expressing RDFunctions, which can be edited, parsed and evaluated. Third, we discuss Tripcel, an implementation of RDFunctions using a spreadsheet metaphor. Using this tool, users can easily edit and execute function expressions and perform analysis tasks on the data stored in an RDF graph.

## 1 Introduction

RDF is a highly generic model for data representation. Its fundamental information unit is the triple, which denotes a specific kind of relationship between two entities (resources), or between an entity and a literal value. As such, it can be used to represent arbitrary kinds of data, as shown by the Linked Data community project[1] and various applications, e.g., in the life sciences field [24] or the Semantic Desktop [25].

RDF data sets are currently accessed and used by applying one of the following two metaphors: either, users directly navigate and browse them using tools that display the graph in a human-perceivable manner (e.g., in tabular form of varying complexity [4, 14]), or using graphical rendering (e.g., in the form of graphs[2] or using rendering template languages [7]). Such tools provide the user with direct, intuitive access to any resource or triple found in the graph, but

[1] Linked Data: `http://linkeddata.org`
[2] RDF-Gravity: `http://semweb.salzburgresearch.at/apps/rdf-gravity`

cannot be reasonably applied to very large graphs. On the other hand, if RDF data are used within applications (i.e., graphs are not directly exposed to the end user), APIs provided by Semantic Web frameworks or query languages (of which SPARQL [23] is the most prominent one) are used. These allow developers to express information needs in a formalized and structured form. However, SPARQL queries can become complex to write and to evaluate [21]; additionally, SPARQL currently lacks certain features that are needed in several use cases (e.g., sub-queries and aggregates).

We are searching for metaphors to interact with Semantic Web data in a way that is intuitive for users, can be quickly authored and evaluated, but still is highly expressive and grounded in a formal model. We were inspired by the concept of *spreadsheets* [18], where data (usually, numbers and text) and calculations (formulas) are arranged in a grid structure of cells, and can be directly inspected and edited by the user. Formulas may refer to the contents of other cells by their coordinates, and evaluation results are displayed immediately after cells have been changed. Although the spreadsheet concept is not free of errors [22] it is a powerful tool and suitable for many application scenarios.

In this paper, we present our approach to merge the spreadsheet concept with the RDF data model. As its formal foundation we present *RDFunctions*, an algebra that consists of mappings between RDF language elements (URIs, blank nodes, and literals). These functions are evaluated not only using parameter values, but also under consideration of the triples stored in an additional *background graph*. Thus, RDFunctions can be used to perform data analysis and computation tasks over RDF data sources. In addition to this formal model, we define a concrete syntax for RDFunction expressions that can be edited by users, and parsed and evaluated by a corresponding engine. Finally, we introduce *Tripcel*, a spreadsheet application that implements the concepts described before, and present an evaluation of its applicability.

## 2 RDFunctions

As the conceptual basis of our approach we define *RDFunctions*, an algebra consisting of mappings between sets of RDF resources. An RDFunction takes a set of RDF elements (i.e., URIs, blank nodes, and literals) as input, and returns another set of RDF elements as result, whereas the evaluation of a function may consider the triples contained in a *background graph*.

A *background graph* $G$ is an RDF graph as defined in [19] and hence consists of a set of triples $< s, p, o >$ which are constructed of elements of the set $\mathbb{U} \cup \mathbb{B} \cup \mathbb{L}$; i.e., URIs, blank nodes, and literals. Following [19] we denote with $universe(G)$ the set of elements that occur in the triples of $G$, and denote by $U_G$, $B_G$, and $L_G$ the sets of URIs, blank nodes, and literals that are elements of this universe, i.e., $universe(G) = U_G \cup B_G \cup L_G$.

An RDFunction $f(\cdot)$ is a mapping $f : \mathcal{P}(\mathbb{U} \cup \mathbb{B} \cup \mathbb{L}) \mapsto \mathcal{P}(\mathbb{U} \cup \mathbb{B} \cup \mathbb{L})$; i.e., it takes a set of RDF elements as input and returns a set of RDF elements as output. These elements must not necessarily be contained in the associated background graph. However, an RDFunction can be defined under consideration of the background graph $G$; in this case we denote the function using an index $f_G(\cdot)$.

The design of this generic function signature is inspired by the fact that when querying data one is not necessarily interested in triples, but in resources or literals (*things*) that fulfill certain criteria: e.g., one might look for resources that fulfil certain criteria, or for literal values of certain properties. Hence we chose to put the RDF language elements (URIs, blank nodes, and literals) instead of triples into the focus of attention.

While an arbitrary number of concrete functions can be defined that fulfil this generic signature, we define in the following a core set of functions that are useful in a broad range of use cases.

**Background Graph Access Functions** We define functions that return groups of elements contained in the background graph; $resources()$, $bnodes()$, $literals()$, and $properties()$, as follows:

$$\begin{aligned} resources_G(\cdot) := \{\ r \ \mid\ \forall s, p, o, r:\ (< r, p, o > \in G \vee < s, p, r > \in G) \\ \wedge\ r \in U_G \cup B_G\} \end{aligned} \tag{1}$$

$$bnodes_G(\cdot) := B_G \tag{2}$$

$$literals_G(\cdot) := L_G \tag{3}$$

$$properties_G(\cdot) := \{\ p \ \mid\ \forall s, p, o: < s, p, o > \in G\} \tag{4}$$

As we can see from these definitions, all background graph access functions discard the input parameter set, i.e., their results are depending only on the triples contained in the background graph.

**Construction Functions** In contrast to background graph access functions, the following group of functions construct RDF elements independent of whether they are contained in the background graph or not, hence the index $\cdot_G$ is not used in their definitions. For the three different forms of literals (plain, typed, and language-tagged) different construction functions are defined[3].

$$resource^{\texttt{uri}}(\cdot) := \texttt{<uri>} \tag{5}$$

$$bnode(\cdot) := \texttt{[]} \tag{6}$$

$$literal^{\texttt{lexicalform}}(\cdot) := \texttt{"lexicalform"} \tag{7}$$

$$literal^{\texttt{lexicalform, uri}}(\cdot) := \texttt{"lexicalform"\^{}\^{}uri} \tag{8}$$

$$literal^{\texttt{lexicalform, lang}}(\cdot) := \texttt{"lexicalform"@lang} \tag{9}$$

Unlike the background graph access functions described before, construction functions ignore the contents of the background graph $G$, as well as the provided input parameter.

[3] We use the triple notation [13] to serialize RDF elements.

**Property Functions** The function $property^p$ returns all values (objects) for property $p$ of the resources given as function parameters ($I$), based on the triples in the background graph $G$.

$$property_G^p(I) := \{\ o \ \mid\ \forall s,p,o : < s,p,o > \in G, s \in I\} \tag{10}$$

One concrete example of such a function is $property^{\texttt{rdfs:label}}$, which would return all labels of the resources given as parameters. Similarly, we define an inverse property function $invproperty^p$ that returns all subjects that have any of the resources given as function parameters as property values (objects) for property $p$:

$$invproperty_G^p(I) := \{\ s \ \mid\ \forall s,p,o : < s,p,o > \in G, o \in I\} \tag{11}$$

**Examples** We illustrate applications of the functions we have defined so far by a number of concrete example. The function

$$invproperty_G^{\texttt{rdf:type}}(I)$$

returns all resources contained in the background graph whose `rdf:type` is one of the resources contained in the input set $I$. Since RDFunctions can be arbitrarily nested, we can use

$$property_G^{\texttt{rdfs:label}}(resources_G(\cdot))$$

to retrieve all `rdfs:label`s from all resources in the background graph (in this case, no input parameters are needed). The function

$$property_G^{\texttt{foaf:name}}(invproperty_G^{\texttt{foaf:knows}}(I))$$

returns the `foaf:name` values of all resources that `foaf:know` any of the resources contained in the input set $I$. Finally, the function

$$property_G^{\texttt{dc:creator}}(invproperty_G^{\texttt{rdf:type}}(resource^{\texttt{swrc:Publication}}(\cdot)))$$

returns the `dc:creator`s of all resources that are typed as `swrc:Publication`.

**Triple Functions** Property functions match a specific property to the predicate position of all triples in the background graph. These functions cover the cases where the property URI is known. To retrieve RDF elements that occur in conjunction with given input resources within a common triple in the background graph whereas the triple's predicate is not known, we define the following functions:

$$objects4subjects_G(I) := \{\ o \mid\ \forall s,p,o : < s,p,o > \in G, s \in I, s \in U_G \cup B_G\} \tag{12}$$

$$subjects4objects_G(I) := \{\ s \mid\ \forall s,p,o : < s,p,o > \in G, o \in I\} \tag{13}$$

$$predicates4subjects_G(I) := \{\ p \mid\ \forall s,p,o : < s,p,o > \in G, s \in I, s \in U_G \cup B_G\} \tag{14}$$

$$predicates4objects_G(I) := \{\ p \mid\ \forall s,p,o : < s,p,o > \in G, o \in I\} \quad (15)$$

Since literals are not allowed in the subject position of RDF triples, the functions $objects4subjects_G(\cdot)$ and $predicates4subjects_G(\cdot)$ consider only those elements of the input set $I$ that are URIs or bnodes, while literals are discarded.

**Aggregate Functions** An aggregate function returns a single value, which is computed from a set of input values. In the context of RDF, certain aggregate functions can be applied to all types of graph elements (e.g., `count()`), while others can be applied only to typed literal values, e.g., `avg()`, `min()`, or `sum()`. RDFunctions can be easily extended by aggregate functions; for the sake of brevity we give here as an example only the definition of the `count()` function that returns the input set's cardinality as typed RDF literal, where $\mid I \mid$ is the cardinality of $I$:

$$count(I) :=\ literal^{|I|,\texttt{xsd:integer}}(\cdot) \quad (16)$$

**Filter Functions** SPARQL provides a mechanism to test the values of RDF elements through the `FILTER` element (cf. [23], Section 11). The RDFunction framework provides the function `filter()` to incorporate SPARQL filter expressions; however the semantics of filter expression evaluation is different in RDFunctions. In contrast to SPARQL, RDFunctions are evaluated not over a graph pattern, but over the set of input elements $I$ (i.e., URIs, blank nodes, and literals). Hence a filter evaluation cannot distinguish between bindings of different variables, as it is the case in graph patterns. Thus, when evaluating the filter expression, *all* variables are bound to the same element (taken from the input set $I$), which effectively implies that an RDFunctions filter function may contain only one variable. This restriction is indicated by the index of the SPARQL `FILTER` function in the definition of the filter function (17), which binds all expression variables to a single element $e$; this binding is indicated by the notion $?{*} = e$ in (17). All input elements for which the filter evaluates to `false` are discarded, and all other elements are added to the function's result set:

$$filter^{\texttt{expression}}(I) := \{e \mid e \in I \wedge \texttt{FILTER}_{?*=e}(\texttt{expression}) \neq \texttt{false}\} \quad (17)$$

**Example** As an example that combines aggregate functions and filter functions, the following function returns the number of resources that have a `foaf:birthday` before September 18, 1979 as follows:

$$count(invproperty_G^{\texttt{foaf:birthday}}(\\ filter^{\texttt{?x < "1979-09-18T00:00:00"\^{}\^{}xsd:dateTime}}(literals_G(\cdot)))) \quad (18)$$

**Discussion** RDFunctions are mappings between sets of RDF elements (URIs, bnodes, and literals) that consider the triples contained in a background graph for evaluation. RDFunctions are designed to be nested in order to formulate more complex mappings. Since they are evaluated against a background graph they can also be considered as query algebra over triples contained therein, and we have shown by a number of examples that they can be used to express complex information needs. Considered as a query language, RDFunctions differs from SPARQL because of the different underlying data model: SPARQL queries are evaluated against an RDF graph and return either a set of variable bindings (for `SELECT` queries) or an RDF graph (for `CONSTRUCT` and `DESCRIBE` queries). RDFunctions, on the other hand, are evaluated against a set of RDF language elements, and also return a set of RDF language elements. Consequently, several SPARQL features (e.g., joins or multi-variable `FILTER` expressions) cannot be represented in RDFunctions. On the other hand, RDFunctions provide several features that can currently only be found in proprietary SPARQL extensions; e.g., aggregate functions, arbitrary expression nesting, or sub-queries. Additionally, RDFunctions are easily extensible, since each function that can be reduced to a mapping between RDF elements can be integrated into the algebra.

The efficiency of expression processing (i.e., query execution) heavily depends on the order in which the elements of a formula are nested. For instance, in expression (18) literal elements are filtered according to their value before the RDF property is evaluated. This expression could be rewritten so that the selection based on the `foaf:birthday` property is conducted before the literal values are tested against the filter, which might lead to a more efficient evaluation depending on the structure of the background graph. However such an optimization depends on the actual implementation as well as knowledge about the underlying background graph and is out of the scope of this formal definition.

## 3 Tripcel: Applying RDFunctions in Spreadsheets

The spreadsheet concept is a powerful, widely-used metaphor for the analysis and processing of data. In essence, a spreadsheet is a collection of *cells* that are arranged in a 2-dimensional grid, the *sheet*. Each cell within a sheet may contain a value or a formula. Formulas are evaluated to return a single result value, which can be reused by other cells as input for evaluation. Cells are usually referred to using a coordinate system where columns are identified by letters, and rows are identified by numbers. The coordinate of a cell is obtained by concatenating its column and row identifiers (e.g., `C17` refers to the cell in column 3, row 17).

Spreadsheets are popular because of a number of reasons. First, they allow users to break down complex calculations into smaller units that are easier to understand. This decomposition is driven by the user, not the machine: it is up to the user to decide whether they prefer to write a single complex formula into one cell, or to split the formula into smaller parts and distribute them across multiple cells. Second, spreadsheets combine formal calculations with user-friendly presentation, since cells can be arranged and formatted according to the user's needs and taste. Finally, spreadsheets provide the possibility to explore and compare different scenarios in a simple manner: a user may change one single value in the sheet, and all other cells are immediately updated.

Based on the idea of spreadsheets, we propose *Tripcel*, a spreadsheet variant that considers not only cells and the values and formulas stored therein, but enriches them with the RDFunctions framework and with background information in the form of an RDF graph. In Tripcel each cell contains an RDFunctions expression, and as described in Section 2 the result of the evaluation of this formula depends not only on the results of other cells, but also on the information stored in the background graph. As a significant difference to traditional spreadsheets, Tripcel cells may evaluate to more than one result value; in fact cells may evaluate to sets of RDF language elements.

Tripcel strictly separates the contents of the spreadsheet (i.e., formulas and expressions) from the contents of the background graph. This means that the same Tripcel spreadsheet can be evaluated over different background graphs without any modification. The connection between the formulas in the spreadsheet and the triples in the background graph is established through the functions that consider the background graph $G$ in their evaluation. All functions defined in Section 2 with an index $\cdot_G$ in their name are such functions.

### 3.1 The Tripcel Formula Syntax

In Section 2 an abstract algebra for functions over RDF language elements has been presented. For concrete applications it is however required to express ad serialize function expressions using a concrete syntax. To represent abstract function formulas we have developed an expression syntax under consideration of the following requirements:

- *Readability.* In spreadsheet-based applications, expressions and formulas are usually directly edited by the user. Hence it is necessary that the syntax of expressions is easily readable, understandable, and editable. This implies that all identifiers (like function names) carry meaningful names, and that the number of special language elements (symbols) is reduced to a minimum.
- *Accordance.* It is difficult for users to remember elements of different languages, which convey similar or equal semantics in different syntaxes. In our context, relevant languages include other spreadsheet expression languages, RDF query languages, as well as common mathematic symbols. Hence, identifiers and special symbols in the Tripcel expression language should be reused from these languages wherever possible.
- *Expressivity.* The language should cover all elements of the RDFunctions algebra, as specified in Section 2; this includes RDF elements (URIs, bnodes, literals) as well as functions. Additionally, the syntax must provide means to specify cell references as function parameters.
- *Unambiguousness.* The language should be easy to parse, and it should be possible to decide which model element a token belongs to.

The syntax for cell formulas is defined using EBNF and is reproduced in Figure 1. It defines five types of expressions; *literals*, *literal references*, *resources*,

```
expression           = literal | literalref | resource | function |
                       cellreference ;
literal              = lexicalform ;
literalref           = '"' lexicalform ;
resource             = '<' { '<' uri '>' | curie } '>' ;
function             = '=' functionname [ '[' functionmodifier ']' ]
                       '(' functionparameter ')' ;
functionname         = propertyfunctionname | externalfunctionname ;
functionmodifier     = expression ;
functionparameter    = expression ;
propertyfunctionname = propertyfunction { '/' propertyfunction } ;
propertyfunction     = propertyname | invpropertyname ;
propertyname         = curie ;
invpropertyname      = '~' curie ;
cellreference        = '=' singlecellreference | enumcellreference |
                       areacellreference ;
singlecellreference  = { 'A'..'Z' } { '0'..'9' } ;
enumcellreference    = singlecellreference { ',' singlecellreference } ;
rangecellreference   = singlecellreference '..' singlecellreference ;
externalfunctionname = alpha ;
```

Fig. 1: Tripcel formula syntax in EBNF. For the sake of brevity we omit the production rules for the symbols `lexicalform` (lexical form of literals), `uri` (URIs [3]), `curie` (compact URIs [5]), and `alpha` (alphabetic characters).

*functions*, and *cell references*. For each type, an example is given in Figure 2. A literal expression textually represents an RDF literal. Every cell resource that does not adhere to one of the special syntactic constructs described in the following is interpreted as literal. The entire content of the cell is used as the literal value, and the literal datatype is guessed by analyzing the textual representation (e.g., a representation consisting only of numeric characters is interpreted as `xsd:integer` literal). If the literal datatype cannot be guessed, `xsd:string` is used as default.

A literal reference is a cell expression that starts with a quotation mark (`"`). In contrast to a literal expression, which is directly translated into a literal value, literal references are interpreted with respect to the background graph: the interpretation of a literal reference is the set of RDF resources that have any property whose (literal) value is equal to the string representation of the literal reference. Hence, Tripcel literal references correspond to the $subjects4objects_G(\cdot)$ function defined in Section 2, where the input set $I$ contains exactly one literal.

A resource can be explicitly instantiated by a cell expression that starts with an opening angle bracket (`<`), followed by a CURIE [5] that identifies the re-

| **Type** | **EBNF Rule** | **Example** |
|---|---|---|
| Literal | `literal` | `ISWC 2009` |
| Literal Reference | `literalref` | `"ISWC 2009` |
| Resource | `resource` | `<dogfood:conference/iswc/2009>`<br>`<<http://www.semanticweb.org>>` |
| Function | `function` | `=filter[isLITERAL(?)](...)`<br>`=filter[?>"M"](...)` |
| Cell Reference | `cellreference` | `=B4`<br>`=A3,B6,F8`<br>`=A5..C8` |

Fig. 2: Examples of Tripcel expressions

source[4] and a closing angle bracket (`>`); alternatively, full URIs can be used by enclosing them into double angle brackets (`<<` and `>>`)[5]. The interpretation of a resource expression is a set that contains exactly one resource, which is identified by the specified URI (this corresponds to the $resource^{\texttt{uri}}(\cdot)$ construction function).

Function expressions refer to other Tripcel functions (cf. Section 2). They consist of the function name, the function modifier expression, and the function parameter expression. The function name is a string that refers to the RDFunction to be used, while the function modifier and the function parameters can be any kind of cell expression, including functions; hence, nested and recursive expressions can be constructed. Function modifiers influence the behaviour of the respective function; for instance, the `filter` function interprets a provided modifier string as SPARQL `FILTER` expression and evaluates its input according to (17).

Because of their importance, a special syntax has been defined for property functions, which are identified by the property's abbreviated URI; an inverse property function is identified by a preceding tilde character. For example, the expression `=rdfs:label(...)` corresponds to the RDFunction $property_G^{\texttt{rdfs:label}}(\cdot)$, while `=~foaf:knows(...)` corresponds to $invproperty_G^{\texttt{foaf:knows}}(\cdot)$. Property functions can be concatenated using a slash character, whereas they are traversed from right to left: the expression `=foaf:name/~foaf:knows(...)` corresponds to $property_G^{\texttt{foaf:name}}(invproperty_G^{\texttt{foaf:knows}}(\cdot))$. This abbreviated syntax allows users to intuitively define property chains, which are often needed in analysis tasks.

[4] We assume that in the context of the Tripcel sheet suitable URI prefixes are defined for all URIs under consideration. For a discussion on potential problems that may arise when URI prefixes are used in user interfaces we refer to [26].

[5] We are aware of the fact that the usage of CURIEs in combination with angle brackets does not correspond to typical RDF serialization formats. However we have chosen this syntax because we want to provide a possibility to enter CURIEs since they are easier to remember, but at the same time need a mechanism to unambiguously identify them as URIs.

Finally, cell references are used to link formulas across different cells. By using a cell reference, the results of the referenced cell(s) are inserted at the point of reference. Currently Tripcel supports three kinds of cell references; single, enumerated, and range. Single cell references are substituted by the referenced cell's result set; for enumerated and range cell references the union of all referenced cells' results is returned. For instance, the enumerated cell reference formula `=B3,B6,C6` will evaluate the formulas in the three specified cells and construct the union of all resulting RDF elements.

## 3.2 Implementation

We have implemented a prototypical spreadsheet application that uses the RDFunctions model and the Tripcel syntax, which have been presented in the previous sections[6]. This tool allows users to load background graphs; to edit, load, and safe Tripcel sheets; and to inspect the evaluation results of each cell in more detail.

The Tripcel application is divided into four layers, reflecting the conceptual components described so far. The basis of the Tripcel application is the *background graph layer*, which is implemented using the Jena Semantic Web framework[7]. All details of RDF storage are hidden by this framework, hence it is in principle possible to connect Tripcel to any RDF data source (e.g., in-memory, database-backed, or remote). However, Tripcel operates on both the Jena Model API and its SPARQL implementation ARQ[8] since several Tripcel functions cannot be efficiently implemented using pure SPARQL; consequently only such data sources can be connected that support both access methods.

On top of the background graph layer the *RDFunctions layer* is situated. This layer implements the semantics of RDFunctions as described in Section 2 in a flexible manner: functions are realized as Java classes that implement a specific interface, thus it is possible to extend this layer by new RDFunctions without modifications to existing code. The RDFunctions layer is responsible for the evaluation of Tripcel formulas; the RDF-specific parts of this layer are likewise implemented using Jena.

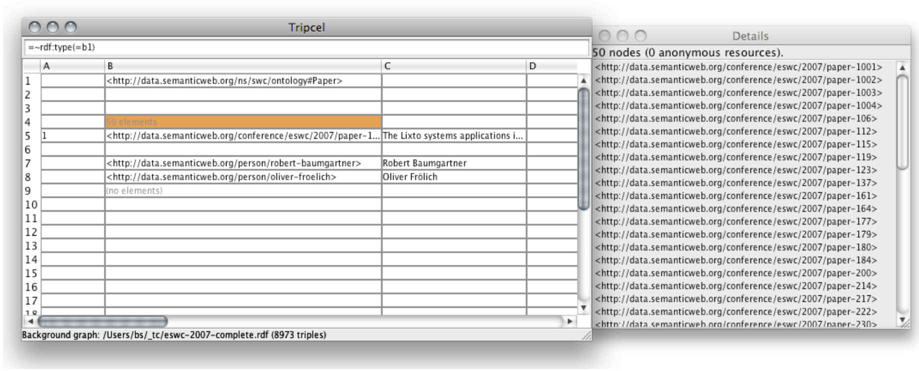
One level above, the *spreadsheet layer* implements the logic of Tripcel spreadsheets. Its responsibility is to manage the contents of cells and their interdependencies. It receives formulas (entered by the user) from the GUI layer (see below), passes them to the RDFunctions layer for evaluation, and buffers the returned evaluation results. It also maintains a cell dependency graph and, upon a cell change, propagates notifications to all depending cells.

Finally, the *GUI layer* provides a graphical representation of a Tripcel sheet (see Figure 3). It renders cells in a grid, provides an editing interface for formulas,

[6] The prototype can be downloaded from `http://www.ifs.univie.ac.at/schandl/2009/06/tripcel`.

[7] Jena Semantic Web Framework: `http://jena.sourceforge.net`

[8] ARQ: `http://jena.sourceforge.net/ARQ`



=~rdf:type(=b1)

| | A | B | C | D |
|---|---|---|---|---|
| 1 | | <http://data.semanticweb.org/ns/swc/ontology#Paper> | | |
| 2 | | | | |
| 3 | | | | |
| 4 | | 50 elements | | |
| 5 | 1 | <http://data.semanticweb.org/conference/eswc/2007/paper-1... | The Lixto systems applications i... | |
| 6 | | | | |
| 7 | | <http://data.semanticweb.org/person/robert-baumgartner> | Robert Baumgartner | |
| 8 | | <http://data.semanticweb.org/person/oliver-froelich> | Oliver Frölich | |
| 9 | | (no elements) | | |
| 10 | | | | |
| 11 | | | | |
| 12 | | | | |
| 13 | | | | |
| 14 | | | | |
| 15 | | | | |
| 16 | | | | |
| 17 | | | | |

Background graph: /Users/bs/_tc/eswc-2007-complete.rdf (8973 triples)

50 nodes (0 anonymous resources).

```
<http://data.semanticweb.org/conference/eswc/2007/paper-1001>
<http://data.semanticweb.org/conference/eswc/2007/paper-1002>
<http://data.semanticweb.org/conference/eswc/2007/paper-1003>
<http://data.semanticweb.org/conference/eswc/2007/paper-1004>
<http://data.semanticweb.org/conference/eswc/2007/paper-106>
<http://data.semanticweb.org/conference/eswc/2007/paper-112>
<http://data.semanticweb.org/conference/eswc/2007/paper-115>
<http://data.semanticweb.org/conference/eswc/2007/paper-119>
<http://data.semanticweb.org/conference/eswc/2007/paper-123>
<http://data.semanticweb.org/conference/eswc/2007/paper-137>
<http://data.semanticweb.org/conference/eswc/2007/paper-161>
<http://data.semanticweb.org/conference/eswc/2007/paper-164>
<http://data.semanticweb.org/conference/eswc/2007/paper-177>
<http://data.semanticweb.org/conference/eswc/2007/paper-179>
<http://data.semanticweb.org/conference/eswc/2007/paper-180>
<http://data.semanticweb.org/conference/eswc/2007/paper-184>
<http://data.semanticweb.org/conference/eswc/2007/paper-200>
<http://data.semanticweb.org/conference/eswc/2007/paper-214>
<http://data.semanticweb.org/conference/eswc/2007/paper-217>
<http://data.semanticweb.org/conference/eswc/2007/paper-222>
<http://data.semanticweb.org/conference/eswc/2007/paper-230>
```

Fig. 3: Tripcel Screenshot: Spreadsheet Window (left) and Detail Window (right)

and displays evaluation results. As Tripcel cells, in contrast to classical spreadsheets, may contain multiple values, the GUI additionally provides a separate detail window where all elements contained in the selected cell are displayed.

The Tripcel application provides interaction mechanisms similar to well-known spreadsheet applications. Normally cells are filled with their evaluation results. If the evaluation of a cell's formula results in more than one element, the number of results (e.g., *"(7 elements)"*) is displayed. When the user clicks on a cell, an editor line is provided where the user can inspect and modify the cell formula. When the user presses enter or selects a different cell, the edited formula is re-evaluated, and changes are propagated to all depending cells.

## 4 Evaluation

**Qualitative Analysis** To estimate the usability and potential impact of our approach, we have performed a qualitative analysis on an initial group of 5 test persons, most of which are experts in the fields of Semantic Web, RDF, and query languages[9]. This analysis consisted of a think-aloud evaluation, followed by a structured questionnaire. In the course of the think-aloud evaluation, all candidates were asked to perform a tutorial of approx. 20 minutes length, during which they would learn the most important features of Tripcel and to get familiar with Tripcel formulas, their syntax, and the results.

During the think-aloud evaluation, the users individually performed the tasks described in a written tutorial, while they were observed by an interviewer. They were asked to immediately tell any thoughts they had during the task completion, regardless of whether they had to do with usability aspects, the entire Tripcel concept as such, or pure implementation issues and bugs. The goal of the think-aloud sessions was to estimate which associations and thoughts were triggered

[9] The material that was used during the evaluation can be downloaded from `http://www.ifs.univie.ac.at/schandl/2009/06/tripcel`.

by Tripcel, and in which aspects the system could be improved. Most issues that were revealed during these sessions regarded the user interface (e.g., visual feedback or the application's general look and feel), missing features (which were structurally collected during the questionnaire, see below), or the syntax and semantics of formula expressions.

After the think-aloud session, each participant was asked to fill a questionnaire that was meant to reflect their impression on the concept and the tool, and to identify potential for improvements in a structured manner. The first part of the questionnaire consisted of questions that were to be answered on a 5-level Likert scale (1 = strongly disagree, 5 = strongly agree) and contained questions addressing Tripcel's general applicability and usability. The second part consisted of open questions addressing the user's impression on specific features as well as potential application fields. Finally, participants were asked to assess their familiarity with Semantic Web technologies and spreadsheets.

Participants rated the usefulness of the tool to get an overview on data with an average of 3.0 ($\sigma = 1.0$) for unknown data, and 3.6 ($\sigma = 1.5$) for known data. The syntax of Tripcel formula expressions was considered to be understandable (3.2, $\sigma = 1.3$) and even better memorizable (4.0, $\sigma = 1.0$). Participants agreed that RDF skills are required to use the tool; its usability for users without RDF skills was denied (1.8, $\sigma = 1.3$). However, for users with RDF skills the usability of the tool was rated very high (4.8, $\sigma = 0.4$). The participants considered themselves to be experts in Semantic Web technologies (RDF: 4.0, $\sigma = 1.4$; SPARQL: 3.8, $\sigma = 1.3$).

In a series of qualitative questions the participants were asked to judge the features of the application. Amongst the positively rated features were GUI aspects like the familiar interaction metaphors and their resemblance to spreadsheet applications, and the fast execution times of formula evaluation. The participants also liked the possibilities and expressivity of the formula language, especially the ability to formulate property paths and aggregate functions. Finally, the ability to apply the same formula sheet to different background graphs was appreciated.

The participants outlined several missing features, including the ability to load multiple RDF documents into one background graph, or to quickly switch between multiple background graphs. The ability to visualize the background graph or cell contents in the form of graphs or charts (as known from spreadsheet tools) was required by several participants. On the GUI level, features like auto-completion and syntax highlighting were mentioned, which would increase the application's usability and reduce the error rate. We are currently in the process of reviewing the detailed requirements for new features, which will be implemented subsequently.

**Quantitative Analysis** As described in Section 3.2, Tripcel has been implemented on top of the Jena Semantic Web framework, and Tripcel functions are implemented using the Jena Model API or the ARQ SPARQL engine, depending on the function type. Consequently, the execution times of such calls and queries are not under the control of our implementation, and additionally depends on the size of the background graph. Here we refer to previous works on performance evaluation of different triple stores (e.g., [10, 6]).

An essential feature of spreadsheets is the possibility to break down complex calculations into smaller units. By resolving a sheet's dependency graph, formulas that are distributed across multiple cells could be merged and optimized before they are translated into queries and executed against the background graph. However the user of a spreadsheet tool expects to be able to inspect intermediate results, which ultimately implies that each formula contained in a cell must be evaluated independently from other formulas. In our Tripcel implementation we follow the approach to buffer evaluation results in-memory for each cell as long as the cell formula (and the formula of any antecedent cell) is not modified. While this approach potentially requires more memory, it significantly reduces the time needed for formula evaluation.

## 5 Related Work

As mentioned before, SPARQL lacks a number of features that are needed in different application scenarios. These deficiencies have been acknowledged by previous works, which led to a number of proposed extensions. A number of these extensions are addressing similar issues as the RDFunctions framework does: Virtuoso SPARQL Extensions[10] or Jena ARQ[11] both provide aggregates and so-called pointer operators that reduce the number of variables needed in triple patterns. However we choose not rely on proprietary language extensions for our implementation. Currently, a number of proposed feature extensions for the next version of the SPARQL language are under review by the W3C SPARQL Working Group.

A number of languages have been proposed that allow to query for more complex triple patterns than it is currently possible using SPARQL. Many of these approaches provide mechanisms to navigate between nodes in an RDF graph, as can be done with the RDFunctions framework. This includes nSPARQL and rSPARQL [2], which provide means to express navigational expressions over RDF graphs, which can be evaluated under consideration of RDFS semantics. RDFunctions currently implements a subset of the features provided by nSPARQL for the sake of simplification. SPARQ2L [1], SPARQLeR [15], and ARQ extend SPARQL with functions for the analysis of path structures in an RDF graph, while the RDF path language of the SILK framework [27] and XsRQL [12] are independent languages. The ability to hierarchical nest property functions relates our work also to the family of RDF path query languages like Versa [20] or RPath [17], which could as well serve as the foundation for Tripcel.

[10] Virtuoso SPARQL Extensions: `http://docs.openlinksw.com/virtuoso/rdfsparql.html#sparqlextensions`

[11] ARQ Extensions: `http://jena.sourceforge.net/ARQ/documentation.html`

Topic Map Query Language [12], although not designed for the RDF model, follows a similar conceptual model. All these approaches provide valuable input for further enhancement and extension of RDFunctions; however we want to ensure that the syntax for RDFunction expressions remains easily to remember.

Another approach comparable to Tripcel are Semantic Web Pipes [16] (which are inspired by Yahoo Pipes[13], an utility for meshing RSS feeds), where RDF data sources can be aggregated and manipulated through linked processing units. Our framework differs from Semantic Web Pipes in that we consider sets of RDF language elements as input and output of functions, rather than RDF graphs.

The interrelationships between spreadsheets and semantic technologies have been studied in a number of works. Tools that are able to extract RDF data from spreadsheets include ConvertToRDF [9], which maps table column headings to ontological concepts, and RDF123 [11], which provides a special language to express the conversion parameters. Other approaches involve the use of GRDDL [8]; examples for this as described e.g., in the GRDDL Primer[14]. Vice versa, since SPARQL `SELECT` query results are tables they can be directly integrated into spreadsheets, as shown e.g., in [26]. However to the best of our knowledge there exists no approach so far that directly integrates processing of RDF language elements into the spreadsheet concept.

## 6 Conclusions and Further Research Directions

In this paper we have presented the concept of RDFunctions, which are mappings between sets of RDF elements under the consideration of background information expressed in an RDF graph. We have defined an extensible conceptual model for RDFunctions, as well as a number of basic functions for processing RDF language elements. These functions have been implemented in the form of Tripcel, a spreadsheet-based tool that allows users to use RDFunctions in order to analyse the contents of RDF graphs. To represent RDFunction expressions we have designed and implemented a formula language which is oriented towards the syntax of popular spreadsheet software. Our approach was evaluated in the course of a study among expert users, who judged Tripcel as being a useful tool for analyzing RDF data, and gave directions for further work.

In the future, we plan to improve the user interface of our implementation and extend it with features that improve usability (e.g., syntax and reference highlighting, auto completion, formula authoring assistants, and more efficient projection of three-dimensional results into the two-dimensional user interface) or that extend functionality (e.g., cell formatting, more advanced aggregate functions, etc.). We aim to extend the range of possible applications of Tripcel by integrating mechanisms that allow the software to connect to multiple remote data sources, which opens the door to evaluate spreadsheets against the Web of Data. Finally, we plan to integrate Tripcel with classical spreadsheet tools

[12] Topic Map Query Language (TMQL): `http://www.isotopicmaps.org/tmql`
[13] Yahoo Pipes: `http://pipes.yahoo.com/pipes/`
[14] GRDDL Primer: `http://www.w3.org/TR/grddl-primer`

in order to facilitate data interoperability. In the first step, we will implement copy+paste functionality; in a second step we plan to implement direct data integration and live synchronization between Tripcel and other spreadsheet software.

**Acknowledgements** Parts of this work have been funded by FIT-IT grants 812513 and 815133 from the Austrian Federal Ministry of Transport, Innovation, and Technology. The author thanks Leo Sauermann for initial inspiration, as well as the participants of the evaluation for their patience and support.

## References

1. Kemafor Anyanwu, Angela Maduko, and Amit Sheth. SPARQ2L: Towards Support for Subgraph Extraction Queries in RDF Databases. In *WWW '07: Proceedings of the 16th International Conference on World Wide Web*, pages 797–806, New York, NY, USA, 2007. ACM Press.
2. Marcelo Arenas, Claudio Gutierrez, and Jorge Pérez. An Extension of SPARQL for RDFS. In Vassilis Christophides, Martine Collard, and Claudio Gutierrez, editors, *SWDB-ODBIS*, volume 5005 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2007.
3. T. Berners-Lee, R. Fielding, and L. Masinter. *Uniform Resource Identifier (URI): Generic Syntax (RFC 3986).* Network Working Group, January 2005.
4. Tim Berners-Lee, Yuhsin Chen, Lydia Chilton, Dan Connolly, Ruth Dhanaraj, James Hollenbach, Adam Lerer, and David Sheets. Tabulator: Exploring and Analyzing Linked Data on the Semantic Web. In *Proceedings of the 3rd International Semantic Web User Interaction Workshop*, 2006.
5. Mark Birbeck and Shane McCarron. *CURIE Syntax 1.0 — A Syntax for Expressing Compact URIs.* World Wide Web Consortium, http://www.w3.org/TR/curie/, 2009. Available at `http://www.w3.org/TR/curie`.
6. Chris Bizer and Andreas Schultz. Benchmarking the Performance of Storage Systems that Expose SPARQL Endpoints. In *Proceedings of the 4th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2008)*, 2008.
7. Pierre-Antoine Champin. Tal4RDF: Lightweight Presentation for the Semantic Web. In *Proceedings of the 5th Workshop on Scripting and Development for the Semantic Web (SFSW2009)*, 2009.
8. Dan Connolly. *Gleaning Resource Descriptions from Dialects of Languages (GRDDL) (W3C) Recommendation 11 September 2007.* World Wide Web Consortium, 2007.
9. Jennifer Golbeck, Michael Grove, Bijan Parsia, Aditya Kalyanpur, and James Hendler. New Tools for the Semantic Web. In *Knowledge Engineering and Knowledge Management: Ontologies and the Semantic Web*, 2002.
10. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158–182, 2005.
11. Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs, and Anupam Joshi. RDF123: From Spreadsheets to RDF. In *The Semantic Web - ISWC 2008*, pages 451–466, 2008.

12. Howard Katz. *XsRQL: an XQuery-style Query Language for RDF (RDF Data Access Working Group Submission)*, 2004. Available at `http://www.fatdog.com/xsrql.html`, retrieved 09-Jun-2009.
13. Graham Klyne and Jeremy J. Carroll. *Resource Description Framework (RDF): Concepts and Abstract Syntax (W3C Recommendation 10 February 2004)*. World Wide Web Consortium, 2004.
14. Georgi Kobilarov and Ian Dickinson. Humboldt: Exploring Linked Data. In *Proceedings of the Linked Data on the Web Workshop (LDOW2008)*, 2008.
15. Krys Kochut and Maciej Janik. SPARQLeR: Extended SPARQL for Semantic Association Discovery. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *ESWC*, volume 4519 of *Lecture Notes in Computer Science*, pages 145–159. Springer, 2007.
16. Danh Le-Phuoc, Axel Polleres, Manfred Hauswirth, Giovanni Tummarello, and Christian Morbidoni. Rapid Prototyping of Semantic Mash-ups through Semantic Web Pipes. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 581–590, New York, NY, USA, 2009. ACM.
17. Keita Matsuyama, Michael Kraus, Kazuhiro Kitagawa, and Nobuo Saito. A Path-Based RDF Query Language for CC/PP and UAProf. *Pervasive Computing and Communications Workshops, IEEE International Conference on*, 0:3, 2004.
18. Richard Mattessich. Budgeting Models and System Simulation. *The Accounting Review*, 36(3):384–397, July 1961.
19. Sergio Muñoz, Jorge Pérez, and Claudio Gutiérrez. Minimal Deductive Systems for RDF. In *The Semantic Web: Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, 2007.
20. Chimezie Ogbuji. Versa: Path-Based RDF Query Language. *XML.com*, 2005. Available at `http://www.xml.com/pub/a/2005/07/20/versa.html`.
21. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and Complexity of SPARQL. In Isabel F. Cruz, Stefan Decker, Dean Allemang, Chris Preist, Daniel Schwabe, Peter Mika, Michael Uschold, and Lora Aroyo, editors, *International Semantic Web Conference*, volume 4273 of *Lecture Notes in Computer Science*, pages 30–43. Springer, 2006.
22. Stephen G. Powell, Kenneth R. Baker, and Barry Lawson. A Critical Review of the Literature on Spreadsheet Errors. *Decision Support Systems*, 46(1):128–138, 2008.
23. Eric Prud'hommeaux and Andy Seaborne. *SPARQL Query Language for RDF (W3C Recommendation 15 January 2008)*. World Wide Web Consortium, 2008.
24. Alan Ruttenberg, Jonathan A. Rees, Matthias Samwald, and M. Scott Marshall. Life Sciences on the Semantic Web: the Neurocommons and Beyond. *Briefings in Bioinformatics*, 10(2):193–204, 2009.
25. Leo Sauermann, Ansgar Bernardi, and Andreas Dengel. Overview and Outlook on the Semantic Desktop. In Stefan Decker, Jack Park, Dennis Quan, and Leo Sauermann, editors, *Proceedings of the 1st Semantic Desktop Workshop*, volume 175, Galway, Ireland, November 2005. CEUR Workshop Proceedings.
26. Bernhard Schandl. Representing Linked Data as Virtual File Systems. In *Proceedings of the 2nd International Workshop on Linked Data on the Web (LDOW), Madrid, Spain*, 2009.
27. Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. Silk – A Link Discovery Framework for the Web of Data. In *Proceedings of the 2nd International Workshop on Linked Data on the Web (LDOW), Madrid, Spain*, 2009.