An Ontology Editor for Defining Cartesian Types to Represent $n$-ary Relations

Christian Willms, Hans-Ulrich Krieger, Bernd Kiefer
German Research Center for Artificial Intelligence (DFKI)
Campus D3 2, Stuhlsatzenhausweg 3, D-66123 Saarbrücken, Germany
christian.willms@dfki.de, krieger@dfki.de, kiefer@dfki.de

Abstract

Arbitrary $n$-ary relations ($n \geq 1$) can, in principle, be realized through binary relations obtained by a reification process which introduces new individuals to which the additional arguments are linked via “accessor” properties. Modern ontologies which employ standards such as RDF and OWL have mostly obeyed this restriction, but have struggled with it nevertheless. In (Krieger and Willms, 2015), we have laid the foundations for a theory-agnostic extension of RDFS and OWL and have implemented in the last year an extension of Protégé, called $\times$-Protégé, which supports the definition of Cartesian types to represent $n$-ary relations and relation instances. Not only do we keep the distinction between the domain and the range of an $n$-ary relation, but also introduce so-called extra arguments which can be seen as position-oriented unnamed annotation properties and which are accessible to entailment rules. As the direct representation of $n$-ary relations abolishes RDF triples, we have backed up $\times$-Protégé by the semantic repository and entailment engine HFC which supports tuples of arbitrary length. $\times$-Protégé is programmed in Java and is made available under the Mozilla Public License.

Keywords: ontology editor, $\times$-Protégé, Cartesian types, $n$-ary relations, RDF, RDFS, OWL, $n$-ary Description Logics.

1. Description Logics, OWL, and RDF

Relations in description logics (DLs) are either unary (so-called concepts or classes) or binary (roles or properties) predicates (Baader et al., 2003). As the designers of OWL (Smith et al., 2004; Hitzler et al., 2012) decided to be compatible with already existing standards, such as RDF (Cyganiak et al., 2014) and RDFS (Brickley and Guha, 2014), as well as with the universal RDF data object, the triple,

\[
\text{subject predicate object}
\]

a unary relation such as $C(a)$ (class membership) becomes a binary relation via the RDF type predicate:

\[
\text{a rdf:type C}
\]

For very good reasons (mostly for decidability), DLs usually restrict themselves to decidable function-free two-variable subsets of first-order predicate logic. Nevertheless, people have argued ver early for relations of more than two arguments (Schmolze, 1989), some of them still retaining decidability and coming up with a better memory footprint and a better complexity for the various inference tasks (including querying) than their triple-based relatives (Krieger, 2012; Krieger, 2014). This idea conservatively extends the standard triple-based approach ($n + 1$ being the arity of the predicate):

\[
\text{subject predicate object}_1 \ldots \text{object}_n
\]

Using a standard relation-oriented notation, we often interchangeably write

\[
p(s, a_1, \ldots, o_n)
\]

Here is an example, dealing with diachronic relations (Sider, 2001), relation instances whose object values might change over time, but whose subject values coincide with each other. For example (quintuple representation),

\[
\text{peter marriedTo liz 1997 1999}
\]

\[
\text{peter marriedTo lisa 2000 2010}
\]

or (relation notation)

\[
\text{marriedTo(peter, liz, 1997, 1999)}
\]

\[
\text{marriedTo(peter, lisa, 2000, 2010)}
\]

which we interpret as the (time-dependent) statement that Peter was married to Liz from 1997 until 1999 and to Lisa from 2000–2010.

In a triple-based setting, semantically representing the same information requires a lot more effort. There already exist several approaches to achieve this (Welty and Fikes, 2006; Gangemi and Presutti, 2013; Krieger and Declerck, 2015), all coming up with at least one brand-new individual (introduced by a hidden existential quantification), acting as an anchor to which the object information (the range information of the relation) is bound through additional properties (a kind of reification). For instance, the so-called $N$-ary relation encoding (Hayes and Welty, 2006), a W3C best-practice recommendation, sticks to binary relations/triples and uses container objects to encode the range information (ppt1 and ppt2 being the new individuals):

\[
\text{peter marriedTo ppt1}
\]

\[
\text{ppt1 rdf:type nary:PersonPlusTime}
\]

\[
\text{ppt1 nary:value liz}
\]

\[
\text{ppt1 nary:starts "1997" xsd:gYear}
\]

\[
\text{ppt1 nary:ends "1999" xsd:gYear}
\]

\[
\text{peter marriedTo ppt2}
\]

\[
\text{ppt2 rdf:type nary:PersonPlusTime}
\]

\[
\text{ppt2 nary:value lisa}
\]

\[
\text{ppt2 nary:starts "2000" xsd:gYear}
\]

\[
\text{ppt2 nary:ends "2010" xsd:gYear}
\]

As we see from this small example, a quintuple is represented by five triples. The relation name is retained, however, the range of the relation changes from, say, Person to the type of the container object which we call here PersonPlusTime.

Rewriting ontologies to the latter representation is an unpleasant enterprise, as it requires further classes, refines property signatures, and rewrites relation instances,
as shown by the marriedTo example above. In addition, reasoning and querying with such representations is extremely complex, expensive, and error-prone. Unfortunately, the former tuple-based representation which argues for additional (temporal) arguments is not supported by ontology editors today, as it would require to deal with general n-ary relations \( (n \geq 2) \). \textit{x-protege} fills exactly this gap.

2. Further Motivation

\textit{x-protege} supports the definition of Cartesian types, composed from standard OWL classes and XSD datatypes. Given Cartesian types and by keeping the distinction between the domain \( \mathbb{D} \) and the range \( \mathbb{R} \) of a binary property \( p \), it is now possible to define \( m + n \)-ary relations \( p \subseteq \mathbb{D}_1 \times \cdots \times \mathbb{D}_m \times \mathbb{R}_1 \times \cdots \times \mathbb{R}_n \). The deeper reason why it is useful to separate domain and range arguments from one another is related to the so-called property characteristics built into OWL, e.g., symmetry or transitivity. This ultimately allows us to generalize the corresponding entailment rules, by replacing atomic arguments of a relation, but also provides a GUI has access to. \textit{x-protege} not only keeps the distinction between the domain and range arguments of a relation, but also provides further distinct \textit{annotation}-like arguments, called \textit{extra} arguments which have been shown useful in various situations and which are accessible to entailment rules of the above kind. Consider a binary symmetric property \( q \) which we would like to generalize by the concept of valid time (the time in which an atemporal statement is true), thus the corresponding entailment rule needs to be extended by two further temporal arguments \( b \) and \( e \):

\[
q(x, y, b, e) \rightarrow q(y, x, b, e)
\]

By assuming that the temporal arguments are part of the domain and/or range of \( q \), we are running into trouble as symmetric properties require the same number of arguments in domain and range position. Thus, we either need to adjust this rule, i.e.,

\[
q(x, b, e, y, b, e) \rightarrow q(y, b, e, x, b, e)
\]

or assume that \( b \) and \( e \) have a special “status”. We decided for the latter and call such information \textit{extra arguments}. As an example, the former marriedTo relation (a symmetric relation) is of that kind, thus having the following relation signature (assuming a biography ontology with class Person):

\[
\begin{array}{c}
\text{Person} \times \text{Person} \times \text{xsd:}\text{gYear} \\
\text{domain} \ \
\text{range} \ \
2 \text{ extra arguments}
\end{array}
\]

Other non-temporal examples of extra arguments might involve space (or spacetime in general), using further XSD custom types, such as point2D or point3D, in order to encode the position of a moving object over time (Keshavdas and Kruijff, 2014).

More linguistically-motivated examples include the \textit{direct} representation of ditransitive and ergative verb frames, including adjuncts (Krieger, 2014). We will present an example of this at the end of Section 7, when defining the quaternary relation obtains. Such kinds of properties are often wrongly addressed in triple-based settings through relation composition, applied to the second argument of the corresponding binary relation. This does not work in general, but only if the original relation is \textit{inverse functional}.

As a last example, we would like to mention the \textit{direct} representation of uncertain statements in medicine or technical diagnosis in an extension of OWL (Krieger, 2016) which is far superior to various encodings described in (Schulz et al., 2014) which have accepted the boundaries of RDF triples in order to be compatible with an existing standard.

3. Protégé, \textit{x}-Protégé, and HFC

Protégé is a free, open source ontology editor, providing a graphical user interface to define and inspect ontologies (http://protege.stanford.edu). Protégé version 4 has been designed as a modular framework through the use of the OSGi framework as a plugin infrastructure (https://www.osgi.org/developer/). For this reason, \textit{x}-Protégé has been implemented as an EditorKitFactory plugin for Protégé, replacing the built-in OWL EditorKitFactory. The EditorKit is the access point for a particular type of model (in our case, a model based on n-tuples) to which a GUI has access to.

\textit{x-protege} is divided into three separate components (Figure 1, large right box). The “bottom” layer is realized by HFC (Krieger, 2013), a bottom-up forward chainer and semantic repository implemented in Java which is comparable to popular systems such as Jena and OWLIM (http://www.dfkj.de/it/onto/hfc/). HFC supports RDFS and OWL reasoning à la (Hayes, 2004) and (ter Horst, 2005), but at the same time provides an expressive language for defining custom rules, involving functional and relational variables, complex tests and actions, and the replacement of triples in favour of tuples of arbitrary length. The query language of HFC implements a subset of SPARQL, but at the same time provides powerful custom \( M:N \) aggregates \((M, N \geq 1)\), not available in SPARQL.

The data read in by HFC is preprocessed and transformed into an \textit{x}-Protégé model. Among other things, it contains inheritance hierarchies for classes and properties which are directly used to visualize the ontology in the graphical user interface of \textit{x}-Protégé.

This GUI consists of several workspaces (similar to Protégé, version 4.3), presenting the ontology itself, the classes, the properties, and the instances. User actions result in an update of the model and HFC’s n-tuple database.

![Figure 1: The three-layered structure of \textit{x}-Protégé.](image-url)
In the next section, we will look into some of these workspaces (or tabs), assuming the marriedTo example from Sections 1. and 2.

4. Class Tab

When starting ×-Protégé the class hierarchy consists of a unique, most general type, called Thing+ in the GUI which subsumes every other Cartesian type and which can be formally defined as

\[
\text{Thing}^+ := \bigcup_{i=1}^{k} (\text{owl:Thing} \sqcup \text{xsd:AnyType})^i
\]

For a given ontology, \( k \) is fixed (finite, of course). Initially, Thing+ has two direct subtypes, viz., owl:Thing and xsd:AnyType. HFC already provides a set of built-in XSD subtypes, such as xsd:Year (Gregorian Year) or xsd:int (4 Byte integers), but also defines non-standard datatypes, such as xsd:monetary. As in a pure OWL setting, owl:Thing and xsd:AnyType are incompatible, but xsd:AnyType is made available under Thing+ in order to define Cartesian types, such as xsd:year \( \times \) xsd:year for the two extra arguments of the marriedTo relation (or even Person \( \times \) xsd:year \( \times \) xsd:year for the sexternary relation \( q \) in Section 2.). This small type hierarchy is depicted in Figure 2.

Note that the non-singleton Cartesian types are highlighted using red colour and that xsd:year \( \times \) xsd:year is correctly classified as a subclass of the Cartesian type xsd:AnyType \( \times \) xsd:AnyType.

5. Property Tab

As in OWL, we distinguish between the property characteristics owl:DatatypeProperty and owl:ObjectProperty. We group these two classes under the super-property Mixed-Property, as we do allow for further “mixed” property characteristics; e.g., properties which are instantiated with an XSD atom in first place or properties with Cartesian domain and range types which are a mixture of OWL classes and XSD types (and thus are neither datatype nor object properties). Since the quaternary relation marriedTo (binary relation plus two extra args) maps URIs onto URIs, it is classified as an object property (remember, the extra args neither belong to the domain nor range of a property).

However, the ternary relation hasAge (binary relation plus one extra arg) is a datatype property as it maps URIs onto XSD ints (the extra arg is the transaction time, the time when the birthdate was entered to HFC); cf. Figure 3.

We complete the overview of the workspace tabs by coming back to Peter and his relation to Liz and Lisa (cf. Section 1.). From the instance tab, we learn about his two marriages and that he is currently 53 years old (see Figure 6).

The symmetry of the marriedTo relation (see Figure 5) further guarantees that Peter is listed in the instance tabs of Liz and Lisa as well.

7. N-Tuples & I/O Formats

As ×-Protégé allows us to deviate from pure binary relations, certain adjustments to the N-triples format (Carothers and Seaborne, 2014) are necessary, especially as extra arguments need to be represented. Assume a quaternary relation obtains between a person and a degree obtained from an educational organization at a specific time:

\[
\text{obtains} \subseteq \text{Person} \times \text{Degree} \times \text{School} \times \text{xsd:date } D \times R_1 \times R_2 \text{ and A}
\]
In order to let the system know of how many arguments the domain, the range, and the extra part of a relation is composed of, we add further length-related information (infix notation):

- obtains rdfs:domain Person
- obtains rdfs:range Degree School
- obtains nary:extra xsd:date
- obtains nary:domainArity "1"×xsd:int
- obtains nary:rangeArity "2"×xsd:int
- obtains nary:extraArity "1"×xsd:int

Notice that the rdfs:range keyword directly above is followed by two classes: Degree and School (=R₁×R₂). Not only is this kind of representation used in the RBox of an ontology, but also in the TBox, e.g.

Degree School rdfs:subClassOf owl:Thing owl:Thing

as

Degree × School ⊆ T × T

is the case. ABox information is also affected by this style of representation, as, for instance

Peter obtains phd stanford "1985" × xsd:date

Besides providing such an (asymmetric) infix representation, × Protégé let the user decide whether a prefix representation is more appropriate for him/her. So, for instance, the last ABox statement above would then become

Peter obtains phd stanford "1985" × xsd:date

We finally like to stress the fact that once one decided to go for a direct representation of additional arguments and reason upon them, queries and rules will usually intermix tuples of different length. For example, in a valid time approach universal information from the TBox and RBox of an ontology is encoded as triples, whereas assertional knowledge will be represented as quintuples (Krieger, 2012); see HFC rule at the end of Section 8.

8. Future Work

Since × Protégé already uses functionality from HFC (see Section 3.), we would like to add further query and rule definition tabs to the next major version of × Protégé to support the construction of HFC queries and rules (see the two examples below).

The query support in × Protégé will ease the definition of SPARQL-like queries in HFC over n-tuples, using keywords such as SELECT, SELECTALL (for the multiply-out mode in HFC in case equivalence class reduction is enabled), DISTINCT, WHERE, FILTER, and AGGREGATE. Depending on the property signatures, × Protégé will then alarm a user if too less, too many, or wrong arguments have been specified in WHERE clauses, FILTER tests, or AGGREGATE functions. This helps to simply the construction of a query such as

\[
\text{SELECT DISTINCT ?partner} \\
\text{WHERE peter marriedTo ?partner ?start ?end} \\
\text{FILTER GreaterEqual ?start "1998" × xsd:gYear &} \\
\text{LessEqual ?end "2005" × xsd:gYear} \\
\text{AGGREGATE ?noOfPartners = Count ?partner}
\]

which computes how many times Peter was married to distinct women between 1998 and 2005. The results of such queries (viz., tables) will also be displayed in this tab.

The rule support will provide means to define, maintain, and extend RDFS, OWL, and custom rule sets. Again, as is the case for queries, clauses, @test, and @action sections of rules in HFC will benefit from checking for the right number of arguments. For instance, the valid time extension of the entailment rule for transitive properties (ter Horst, 2005) in HFC looks as follows (Krieger, 2012):

\[
?p rdf:type owl:TransitiveProperty // triple \\
@test // 3 LHS tests \\
?x != ?y \\
?y != ?z \\
IntersectionNotEmpty ?start1 ?end1 ?start2 ?end2 // 2 RHS actions \\
@action // new RHS variable \\
?start = Max2 ?start1 ?start2 \\
?end = Min2 ?end1 ?end2 // new RHS variable
\]

In both cases, we would also like to provide a completion mechanism for properties and URIs, as well as for external tests (see @test above) and value-returning functions (see @action above), an extremely useful functionality known from programming environments.

Our ultimate goal is thus to offer × Protégé as a front-end GUI for ontology-based systems, based on HFC.

9. Download

× Protégé version 1.0 as of Monday Feb 15, 2016 can be downloaded from https://bitbucket.org/cwillms/x-protege/downloads/ and is made available under the Mozilla Public License. Here, you will also find a preliminary version of the user guide.

10. Acknowledgements

The research described in this paper has been partially financed by the European project PAL (Personal Assistant for healthy Lifestyle) under Grant agreement no. 643783-RIA Horizon 2020. This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health. We would like to thank the three reviewers for their detailed and useful suggestions.
11. Bibliographical References


