Sequences in Protégé OWL

Nick Drummond[1], Alan Rector[1], Robert Stevens[1], Georgina Moulton[2],
Matthew Horridge[1], Hai H. Wang[1], Julian Seidenberg[1]

1. Bio Health Informatics Group, School of Computer Science, The University of Manchester, UK
2. Northwest Institute for Bio Health Informatics, Manchester, UK
nick.drummond@cs.manchester.ac.uk

ABSTRACT: Sequences are a natural part of the world to be modeled in ontologies. Yet the Web Ontology Language, OWL, contains no built in support specifically for sequences or ordering. It does, however, have constructs that can be used to model many aspects of sequences, albeit imperfectly. This paper describes a design pattern for modeling order using existing OWL-DL constructs. These constructs allow us to use standard DL reasoning to perform pattern matching akin to regular expression matching. Although clearly not the most efficient mechanism, pattern matching with standard DL reasoners works surprisingly well and brings real benefits to users by allowing them to work at a higher level of abstraction than raw sequences and to deal with situations in which the details of the sequences are under specified.

Introduction

OWL has no inbuilt support for ordering. However, the world to be modeled in ontologies expressed in OWL is full of sequences:

- Time related events – *e.g.* sequences of sub-processes, plans, life-stages etc.
- Physically linked structures – *e.g.* Protein sequences and other macromolecules, carriages in a train, etc.
- Conceptually linked structures – *e.g.* documents, data structures, travel itinerary etc.

Unfortunately, the natural constructs from the underlying RDF vocabulary – `rdf:List` and `rdf:nil` – are unavailable in OWL-DL because they are used in the RDF serialization of OWL[1]. Although `rdf:Seq` is not illegal, it depends on lexical ordering and has no logical semantics accessible to a DL classifier. Despite these limitations, we have strong reasons for wanting to express and reason with sequential constructs in OWL-DL.

- *Expressivity* – OWL-DL includes constructs such as transitive properties, which allow more of the semantics of sequences to be represented explicitly than in RDF or OWL-Lite.
- *Reasoning* – DL reasoners can be used to check consistency and infer subsumption – *e.g.* to confirm that a sequence of amino acids contains only amino acids or to infer that a class of sequences is subsumed by another class of sequences – *i.e.* that one pattern is a subset of another pattern.

Our example is that of sequences of amino acids, although the patterns are applicable to many domains. An example ontology is available along with a more general demonstration on the co-ode wesite[2].

Modelling lists as data structures

To model lists as data structures, we follow the standard pattern for linked lists in which each item is held in a "cell" (`OWLList`); each cell has contents ("head") and a pointer to the next cell ("tail"); and the end of the list is indicated by a terminator (`EmptyList`) which also serves to represent the empty list. In RDF these constructs are

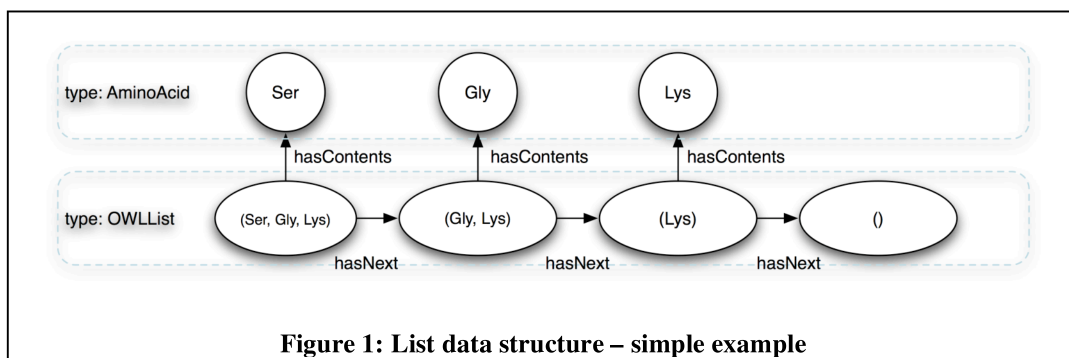

Figure 1: List data structure – simple example

[1] http://www.w3.org/TR/owl-semantics/mapping.html#rdf_List_mapping
[2] http://www.co-ode.org/ontologies/lists/

implemented using the class `rdf:List` for the cell, the individual `rdf:nil` as the terminator, and the two properties `rdf:first` and `rdf:next` for the contents and pointer to the next cell respectively.
However, because we cannot use the RDF vocabulary in OWL-DL we must define our own (Figure 2), an example of which is shown diagrammatically in Figure 1. Whereas the semantics of the properties `rdf:first` and `rdf:next` are implicit in RDF, in OWL we can express more. We want each cell to have exactly one contents item and one next cell, and we want to represent the notion of being a member of the list. This can be done by making `hasContents` and `hasNext` functional, and by defining a transitive property, `isFollowedBy`, as a super-

```
Class(OWLList partial
        restriction(isFollowedBy allValuesFrom(OWLList)))
Class(EmptyList complete
        OWLList
        restriction(hasContents maxCardinality(0)))
EquivalentClasses(EmptyList
                  intersectionOf(OWLList
                                 NOT restriction(isFollowedBy SOME owl:Thing)))
ObjectProperty(hasListProperty domain(OWLList))
ObjectProperty(hasContents Functional super(hasListProperty))
ObjectProperty(hasNext Functional super(isFollowedBy))
ObjectProperty(isFollowedBy Transitive super(hasListProperty) range(OWLList))
```

Figure 2: OWL vocabulary for lists as data structures in concrete abstract syntax

property of `hasNext` as shown. Since this means that `hasNext` implies `isFollowedBy`, any sequence of entities linked by `hasNext` will be inferred to be a chain linked by `isFollowedBy`.

In other words the members of any list are the contents of the first element plus the contents of all of the following elements. The intention is that cells should be directly linked by the functional property `hasNext`. The transitive superproperty, `isFollowedBy,` is typically used in definitions and queries. An example of a fully specified list is shown in Figure 3.

Note that we are representing classes of lists rather than individual lists. We treat classes of lists as patterns. We then use the reasoner to determine which other classes of lists and/or individual lists are subsumed by – *i.e.* match – those patterns.
For uniformity we have chosen to create a class of empty lists, which have neither content nor following members. (The negated existential restriction is used with the property `isFollowedBy` rather than the apparently simpler `cardinality(0)`, because cardinality constraints are not permitted on transitive properties[3]). Note that, from the definitions and equivalence axioms given, we can infer that any list that provably has no contents can have no following elements and *vice versa*.

```
OWLList AND
      hasContents SOME Ser AND
      hasNext SOME (
           OWLList AND
               hasContents SOME Gly AND
               hasNext SOME (
                   OWLList AND
                       hasContents SOME Lys AND
                       hasNext SOME EmptyList))
```

Figure 3: Example of a class of OWL Lists of the form (Ser, Gly, Lys) in simplified syntax

Types of lists

Possible constructs are shown in Figure 4 along with a simplified syntax and examples – there is not enough space to describe the OWL for each, but there is at least one example of each in the demo ontology. We use a sugared shorthand syntax for lists that should be intuitive given the definitions and examples. Space does not permit an exhaustive enumeration, but it is clear that constructs supported are similar in expressivity to that available in regular expressions. Below we describe some of the properties of lists modeled using this pattern:

- The elements are classes, which may be fully defined, e.g. “tiny polar amino acid” or “large charged amino acid”. A defined class can be considered as an implied disjunction of its subclasses. This would equivalent to being able to name a disjunction[4] in a regular expression. Most regular expression languages do not support the use of named subexpressions. Even if named subexpressions were supported, the disjunction would have to be enumerated manually in advance. By contrast, in OWL, the classifier can infer the subclass hierarchy based on

[3] http://www.w3.org/TR/owl-ref/#OWLDL
[4] In regular expression parlance, an “alternation”, usually written [P1 P2 P3] where each Pi is itself a regular expression.

	terminology	meaning	examples
1	(A, B, C)	Exactly ABC (terminated)	abc
2	(A!)	A list consisting only of As	aaa, aa, etc.
3	(A, B, C, …)	Starting with ABC (non-terminated)	cbc, abcx
4	(…, A, B, C)	Ending With ABC (terminated)	abc, xabc
5	(…, A, B, C, …)	Containing ABC	abc, xabc, xabcx, abcx
6	(A*B)	A string of As followed by B	ab, aaab,
7	([A, B, C], B, C)	A or B or C, followed by B then C	abc, bbc, cbc
8	(hasProp some X, B, C)	Restriction followed by B then C	Any abc where a hasProp x
9	¬(A, B, C, …)	Not starting ABC	cbaxx
10	((A, B, C, …), (D, E, F, …))	Starting ABC, followed by anything, followed by DEF, followed by anything	abcdef, abcxxdefx
11	(A, B, C, …) AND (…, A, B)	Starting ABC, and ending AB	abcab, abcxxab
12	()	Empty list (nil)	

Figure 4: Examples of the constructs that can be expressed in OWL Lists

the properties of the amino acids. Different abstractions over the same amino acids can be used for different problems.

- The notion of "0 or more As" or "1 or more As" cannot be expressed on its own without including the terminating pattern or item, even if this is simply the empty list as this requires a recursive definition in OWL.
- There is no way of stating "n As", *i.e.* n repetitions of A, other than by explicitly expanding the list with n occurrences of A.
- It is possible to assert a class of lists in a conjunction, or more generally a boolean combination, of classes defined using the above patterns – as in line 11. However, care is required, as this can give unexpected results. For example, the class of lists that starts with "ABC" and ends with "BCD" is different from the class of lists starting with "ABC" followed by "BCD". That is:
  (A, B, C, ...) AND (..., B, C, D) subsumes (A,B,C,D)
  whereas (A, B, D, ... ,B, C, D) does not.
- There is no way to define a class of "lists" so that it excludes cycles. (Note that the class of lists (A, B, A) does not imply a cycle, merely a list beginning with an A, followed by a B, followed by another (possibly the same) A. Both individual lists (a1, b, a2) and (a1, b, a1) satisfy this definition.
- There is no way to define the class of lists of a specific length except by exhaustively representing the member classes. A more compact form would require the use of cardinality constraints on isFollowedBy, which is transitive. Cardinality constraints on transitive properties are excluded from OWL-DL[5].
- There is no way to define a class of "lists" so that it excludes additional branches being defined using isFollowedBy instead of its functional sub-property hasNext.
- It is not possible to represent a class of lists in which one named sublist *directly follows* another named sublist without an intervening element (as in pattern 10). This can be done only by explicitly re-representing the concatenation of the two patterns as a single list. To make this practical, a macro like mechanism would be required in the tools.

Using OWL lists

The mechanisms described have been used with biologists to capture notions that they would otherwise find difficult to express. Biologists find the ability to work with under-specified sequences and to consider abstractions over sequences useful. Biologists form amino acid patterns (motifs) by looking at collections of similar proteins and deducing that all these proteins have either Arginine or Lysine at this position. A regular expression is then made that captures this inference. With OWL lists, we can capture this notion using pattern 7 or we can enable is a greater abstraction, "large and positive", using pattern 8:

Pattern 7 element: Arg or Lys

Pattern 8 element: AminoAcid and ((hasSize some Large) and (hasCharge some Positive))

The abstraction is more expressive than the disjunction and by underspecifying we may also find further matches that suggest the disjunction is over-constrained. In addition, finding that one under-specified pattern subsumes another has intriguing biological possibilities.

We have implemented a series of wizards and tools in Protégé-OWL to make the construction of the required subset of patterns possible and made the user interface practical and at least partly hide the raw OWL syntax. Further tools to help users formulate and display these notions and to make the pattern easily and completely generic are in progress.

[5] and OWL 1.1

For the test-cases that have been run, results are surprisingly fast, although we would like to investigate further whether some constructs have a greater effect on the reasoning speed than others. The example on the web includes reasonably complex, although intentionally short, classes of lists and classifies on a moderately fast laptop, using Protégé-OWL connected to FaCT++[6] or Pellet[7], in approximately 2 seconds. The accompanying fingerprint example, which models real biological data, uses pattern 10 to "join" six motifs together in sequence. Each motif matches a pattern of approximately 20 elements, with a number of alternative elements at each position. Running through Pellet took between 80-170s to correctly classify various test proteins up to 450 elements long. Because of the recursive nature of the definitions, the main practical difficulties witnessed were that of stack size within editing and reasoning software which can be easily resolved with careful programming.

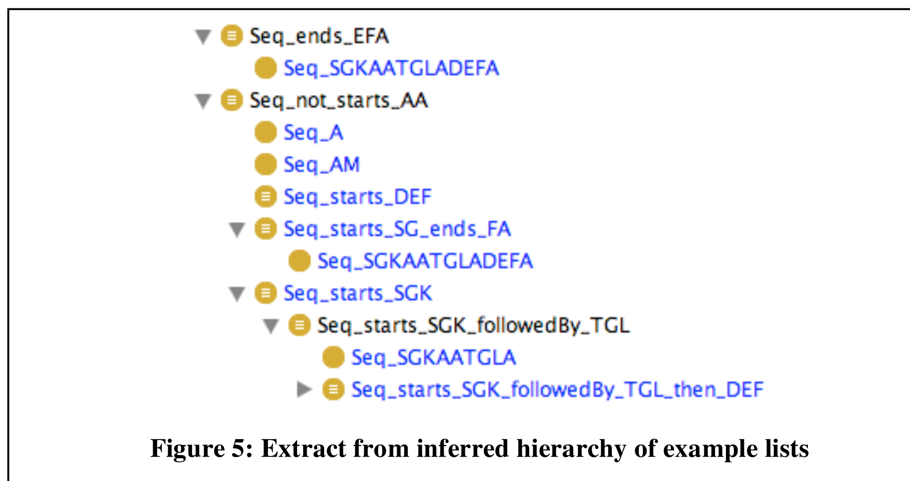

Figure 5: Extract from inferred hierarchy of example lists

Conclusion

In summary, we have described a design pattern for describing sequences in OWL-DL. There are alternatives to be investigated, such as the idea of directly linking elements together without any intervening structure, but the ontological differences between these approaches lie outside the scope of this paper.

Standard reasoners can be used to infer subsumption corresponding to pattern matching over classes of lists and to recognize lists of individuals as belonging to given classes, *i.e.* to treat classes of lists as patterns and to determine whether other classes are more specialized patterns and whether lists of individuals match those patterns. For illustration, a small extract of the classified online example is shown in figure 5. The use of tableaux reasoners, ensures that our matches are sound and complete. However, users must take care to provide complete definitions including both disjointness and closure axioms. Algorithms based on deterministic finite automata as in standard regular expression matchers would almost certainly be faster but the main purpose of using a tableaux reasoner has been to express the list structures along with other notions in a single representation and use reasoners already integrated with OWL-DL.

The most important result of this work is our experience that representing and reasoning over classes of ordered structures in OWL-DL is useful. It provides users with new ways to organise and browse their knowledge at a higher level of abstraction than would otherwise be possible. We have used real examples from protein sequences in biology as a motivation, but the notions are general and can be applied to other notions that are intrinsically ordered.

[6] http://owl.man.ac.uk/factplusplus/
[7] http://www.mindswap.org/2003/pellet/