Putting OWL in Order: Patterns for Sequences in 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 modelled in ontologies. Yet the Web Ontology Language, OWL, contains no specific support for ordering. It does, however, have constructs that can be used to model many aspects of sequences, albeit imperfectly. This paper demonstrates a design pattern for modeling order within OWL-DL. This allows us to use standard DL reasoning to perform pattern matching akin to regular expression matching and works surprisingly well. The main point of this paper is that formulating sequences in OWL-DL 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.

1 Introduction

The world to be modelled is full of sequences:

- Time related events – *e.g.* sequences of sub-processes, biological lifecycles etc.
- Physically linked structures – *e.g.* protein sequences, carriages in a train, etc.
- Conceptually linked structures – *e.g.* book chapters, recipes, travel itinerary etc.

However, not only does OWL have no support for ordering, but the natural constructs from the underlying RDF vocabulary – `rdf:List` and `rdf:nil` – are unavailable in OWL-DL because they are used in its RDF serialization[1]. Although `rdf:Seq` is not illegal, it depends on lexical ordering and has no logical semantics accessible to a DL classifier. Attempts have been made to implement sequences by modeling directly in OWL itself – e.g. the OWL-S (OWL-Services) specification requires order to describe service composition and provides an implementation of lists[2] however this is little more than the RDF vocabulary mirrored in OWL without further semantics.

Despite these limitations, we have strong reasons for wanting to express and reason with sequential constructs in OWL-DL.

[1] http://www.w3.org/TR/owl-semantics/mapping.html#rdf_List_mapping
[2] http://www.daml.org/services/owl-s/1.2/generic/ObjectList.owl

- *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. OWL also allows modeling at multiple levels of *abstraction*, such that sequences can be characterized by their general or more specific properties.
- *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 and to infer that a sequence is subsumed by another – *i.e.* that it matches the pattern represented by that sequence.

The idea of reasoning with sequential structures in OWL-DL using a tableaux reasoner may surprise some readers but this area has been investigated before. Hirsh & Kudenko [1] used DLs to model strings as suffix trees and subsumption to solve substring operations. However, their representation requires extensive rewriting, the relation of the resulting structures to the original lists is not intuitive and, more importantly, the resulting structures grow as the square of the length of the list. This is a serious scaling problem in our domain using lists with hundreds of elements.

We will describe a general list pattern – an intuitive approach related to that suggested by Hayes[3] and incorporated in the Semantic Web Best Practice Working Group's note on n-ary relations [2]. We review the range of constructs possible using this approach, describe their benefits and limitations and we will examine them in the context of a real example from biology, protein sequences, which must be quickly introduced first.

2 Example Application – A Short Introduction to Proteins

Proteins are made up of sequences of amino acids (actually amino acid residues, but we'll keep this simple) and are fundamental to biology.

Biologists also talk more generally about proteins that have various attributes derived from the patterns of amino acids they contain. This is actually a very important part of biochemistry – matching specific proteins to common patterns.

Motif (6PFRUCTKNASE motif 1):
[IV]-A-[VI]-F-D-A-T-N-[TS]-T-[RK]-[EDK]-R-R-[HSDARK]

Protein (F26_YEAST Fructose-2,6-biophosphatase):
MGYSTISNDNDIKVCVIMVGLPARGKSFISQKIIRYLSWLSIKAKCFNVGNYRRDVSGNVPMDA
EFFNFENTDNFKLRELAAQNAIKDIVNFFTKEDGS**VAVFDATNSTRKRRK**WLKDICEKN
NIQPMFLESWSNDHELIINNAKDIGSTSPDYENSEPHVAEADFLERIRQYERFYEPLDPQKDK
DMTFIKLVNIIEEVVINKIRTYLESRIVFYVMNIRPKPKYIWLSRHGESIYNVEKKIGGDSSLSERG
FQYAKKLEQLVKESAGEINLTVWTSTLKRTQQTANYLPYKKLQWKALDELDAGVCDGMTYEEI
EKEYPEDFKARDNDKYEYRYRGGESYRDVVIRLEPVIMELERQENVLIITHQAVLRCIYAYFMN
VPQEESPWMSIPLHTLIKLEPRAYGTKVTKIKANIPAVSTYKEKGTSQVGELSQSSTKLHQLLND
SPLEDKF

Fig. 1. General motif plus matching concrete motif in protein sequence

Proteins can be thought of as containing "motifs", short sequences of amino acids that perform a particular function. Concrete motifs (shown in bold in the Protein sequence in Fig. 1) are usually specified in terms of specific amino acids which are extracted by special search engines, *e.g.* INTERPRO [3]. Biologists often find that within motifs, alternative amino acids can be found at certain points. These generalizations are based on evidence of alternative concrete motifs that perform the same function – biologists then create abstracted motifs that appear more like simple regular expressions (top of Fig. 1).

The twenty individual kinds of amino acids can be categorized along many different axes including size, polarity, charge, etc. In OWL, of course, it is possible to describe amino acids by these features. Taking some examples of substitutions that biologists make in a particular motif, we can spot that often the alternatives share common attributes. We can then express our motif in an even more general form. Fig. 2 shows a smaller example [4] with the 3 levels of abstraction; 2 concrete sequences of three specific amino acids that perform the same function; then a more generalized motif in a form commonly used by biologists; then a more abstract motif expressible in OWL using the amino acid features – "first a tiny polar amino acid, followed by any amino acid, then a large positively charged amino acid".

Such patterns of amino acids are a key to characterizing protein sequences. One of our goals is to allow scientists to explore relationships among proteins characterized by the motifs they contain. To do so, we describe sequences at the class level and then use the DL reasoner to arrange them into subsumption hierarchies. A second goal is to allow scientists to work with incomplete information. For example, a scientist might only know that a sequence consisted of one tiny, polar amino acid, followed by any amino acid then by a large positively charged amino acid. Viewed in this way, we describe such sequences as "underspecified".

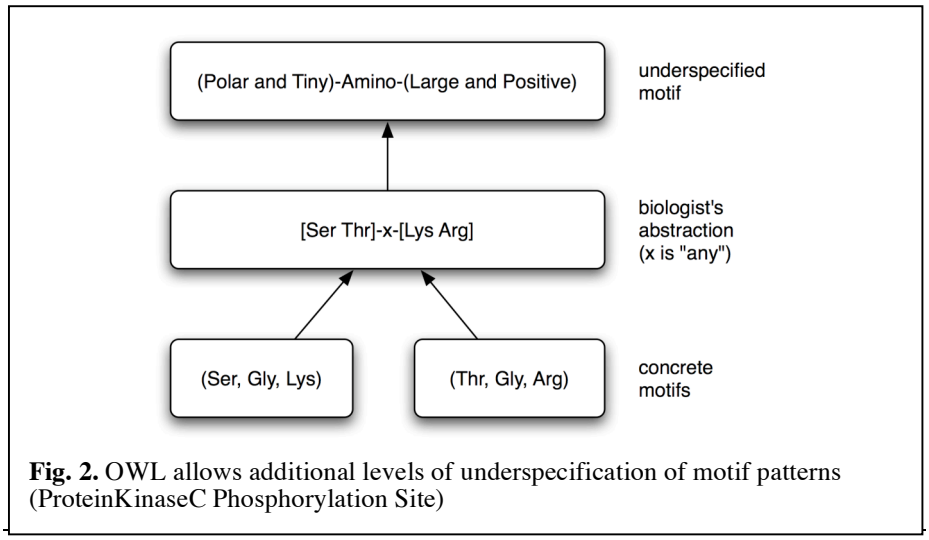

Fig. 2. OWL allows additional levels of underspecification of motif patterns (ProteinKinaseC Phosphorylation Site)

[3] Personal communication, Pat Hayes, 2004

[4] Note that we are modeling our lists at the class level, so for the rest of the paper we use the word OWLList to denote a "class of lists"

3 Describing Sequences in OWL

We follow a standard software engineering pattern for linked lists (Fig. 3) in which each item is held in a "cell" (`OWLList`); each cell has 2 pointers, one to a head (hasContents) and one to the tail cells (hasNext); the end of the list is indicated by a terminator (`EmptyList`) which also serves to represent the empty list. In RDF these constructs are 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.

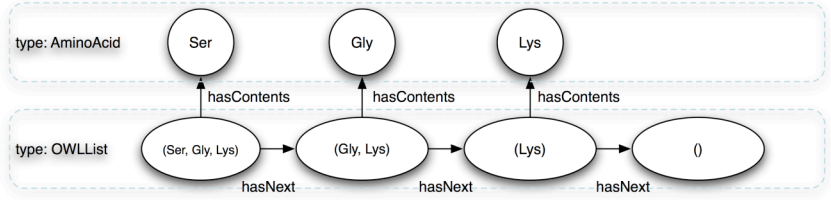

Fig. 3. List data structure – simple example

However, as already mentioned, we cannot use the RDF vocabulary in OWL-DL. Therefore, we must define a separate OWL vocabulary[4] (shown in concrete abstract syntax[5] in Fig. 4), an example of which is shown diagrammatically in Fig. 3. 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-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, in order, for example, to infer that Serine (followed by anything) followed by Argenine subsumes the fully specified sequence Serine, Glysine, Argenine. Alternatively, `isFollowedBy` can be used to indicate incomplete information, for example, that we know that one motif follows another, but not the details of the intervening sequence.

[5] http://owl.man.ac.uk/2003/concrete/latest/

An example of a fully specified list is shown in Fig. 5. In this and subsequent examples we use the simplified "Manchester Syntax"[6] used in the Protégé-OWL plugin[7].

```
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
        super(hasListProperty)Functional)
ObjectProperty(hasNext
        super(isFollowedBy) Functional)
ObjectProperty(isFollowedBy
        super(hasListProperty) Transitive range(OWLList))
```

Fig. 4. OWL vocabulary for lists as data structures

```
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))
```

Fig. 5. Example of an OWLList of the form (Ser Gly Lys) in simplified Manchester syntax

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[8]). Note that,

[6] http://www.co-ode.org/resources/reference/manchester_syntax/
[7] http://protege.stanford.edu/overview/protege-owl.html
[8] http://www.w3.org/TR/owl-ref/#OWLDL

	form	meaning	examples
1	(A, B, C)	Exactly ABC (terminated)	abc
2	(A*)	A list consisting only of As	aaa, aa (or empty)
3	(A, B, C, …)	Starting with ABC (non-terminated)	abc, abcx
4	(…, A, B, C)	Ending With ABC (terminated)	abc, xabc
5	(…, A, B, C, …)	Containing ABC	abc, xabc, xabcx, abcx
6	(A+, B, …)	One or more 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	not(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)	

Fig. 6a. OWLList expressivity and notation summary

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*.

This formulation leaves open the question of whether there is more than one empty list. If we wish to specify that there is a unique empty list, then we must add a further axiom to state that EmptyList is equivalent to the nominal that consists of just that unique individual, *i.e.*

```
EquivalentClass(EmptyList, oneOf(emptyList))
```

However, since not all classifiers handle nominals well, we shall omit this step. It matters here only that we can infer when a class or individual list, is empty.

1.1 Expressivity

Possible constructs are shown in Fig. 6a along with a simplified syntax and examples. Sample class definitions illustrating the patterns are given in Fig. 6b. We use 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 with two important advantages:

- 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[9] in a regular expression. Most regular expression languages do not

[9] In regular expression parlance, an "alternation", usually written [P1 P2 P3] where each Pi is itself a regular expression.

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

- It is possible to assert a list in a conjunction, or more generally a boolean combination, of classes defined using the above patterns – as in pattern 11. However, care is required, as this can give unexpected results. For example, lists that start with "ABC" and end with "BCD" are different from lists starting with "ABC" and 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.

	form	class definition using pattern
1	`(A, B, C)`	as per Fig 5

2	`(A*)`	`List_only_As → List AND`
		`  hasContents ONLY A AND`
		`  isFollowedBy ONLY`
		`  (List AND hasContents ONLY A)`

3	`(A, B, C, …)`	`List_starts_ABC → List AND`
		`  hasContents SOME A AND`
		`  hasNext SOME (List AND`
		`  hasContents SOME B AND`
		`  hasNext SOME (List AND`
		`  hasContents SOME C))`

4	`(…, A, B, C)`	`List_ends_ABC → List_ABC OR`
		`  isFollowedBy SOME List_ABC`
		(where List_ABC follows definition 1)

5	`(…, A, B, C, …)`	`List_contains_ABC → List_starts_ABC OR` `  isFollowedBy SOME List_starts_ABC`

6	`(A+, B, …)`	`List_AsFollowedByB → List AND`
		`  hasContents SOME A AND`
		`  hasNext SOME (`
		`  (List AND (hasContents SOME B)) OR`
		`  List_AsFollowedByB)`

7	`([A|B|C], B, C)`	as per def 1 but substitute `(A OR B OR C)` for `A`
8	`(hasProp some X, B, C)`	as per def 1 but substitute `(hasProp some X)` for `A`
9	`not(A, B, C, …)`	`List_notStarts_ABC → List AND` `  NOT (List_starts_ABC)`

10	`((A, B, C, …),`
	`  (D, E, F, …))`
		`List_starts_ABC_followedBy_DEF →`
		`  List_starts_ABC AND`
		`  isFollowedBy SOME List_starts_DEF`

11	`(A, B, C, …) and` `  (…, A, B)`	`List_starts_ABC_ends_AB →` `  List_starts_ABC AND List_ends_AB`
12	`()`	as per Fig 3

Fig. 6b. Example definitions for OWLList Patterns

Compared to RDF lists, OWL allows much tighter control over the constructs used to describe sequences. However, OWL-DL is not currently sufficiently expressive to exclude all unintended constructs:

- Most importantly, OWLLists cannot exclude additional branches being defined using `isFollowedBy` instead of its functional sub-property `hasNext`. To do so

would require being able to define `isFollowedBy` as the transitive closure of `hasNext` rather than merely being implied by `hasNext`:
i.e.
`hasNext o hasNext o … o hasNext ←→ isFollowedBy`[10]
rather than
`hasNext o hasNext o … o hasNext → isFollowedBy`
OWL only includes the second, weaker, of these statements directly through transitivity, because reasoner optimizations currently only exist for the simple implication [5] and not for the bi-implication.

- OWLLists cannot exclude cycles. To do so would require additional constructs such as being able to declare the `isFollowedBy` property to be antisymmetric. This is ongoing work within the DL community and the issues for optimizing reasoners have not yet been resolved[11]. In the absence of a full logical check, checking for cycles must be done separately. Note that `(A, B, A)` does not imply a cycle, merely a list beginning with an `A` and ending with (possibly the same) `A`. Both individual lists `(a1, b, a2)` and `(a1, b, a1)` satisfy this definition.
- There is no way to define an OWLList 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 restrictions on transitive properties are excluded from OWL-DL.
- The notion of "0 or more `A`s" or "1 or more `A`s" cannot be expressed on its own without including the terminating pattern or item, even if this is simply the empty list. Note also the recursive definition required for this construct in pattern 6.
- Representing an OWList in which one named sublist *directly follows* another named sublist without an intervening element can only be done by explicitly re-representing the concatenation of the two patterns as a single list – pattern 10 cannot enforce this. To make this practical, a macro like mechanism would be required in the tools.
- We cannot use the reasoner to find the minimal set of classes that are used in a given list. i.e. there is no way to define a class such that it subsumes just those classes mentioned in an OWLList, *i.e.* from the definition `L = (A, B, C, A, C)` to define a class `M` that subsumes precisely the classes `A, B,` and `C`. This would have to be done directly by the tools

Within the limitations noted above, the inferences are, as ever with tableaux reasoners, sound and complete. However, users must take care to provide complete definitions including both disjointness and closure axioms. Additional constraints in the tools are required to aid the user in these cases if they are to be assured of the expected results.

[10] The role inclusion (o) syntax specifies a chain of individuals related along the properties given, so the above means eg. a hasNext b hasNext c hasNext d ←→ a isFollowedBy d
[11] Personal communication, Ulrike Sattler, May 2006.
[12] http://www.co-ode.org/ontologies/lists/

4 Results

Two example ontologies are provided on the web for illustration[12]. The motif example contains a set of dummy definitions for protein sequences in order to cover all of the constructs described in Fig. 6, such that the expected subsumptions can be demonstrated. The second, fingerprint example, which models real biological data, uses pattern 10 to "join" six motifs together in sequence. Each motif contains a pattern of approximately 20 elements, with a large number of alternative elements at each position. The fingerprint can then be used to match any sequence that contains all of the given motifs in the given order. For both these examples, computation is surprisingly fast, although we would like to investigate further whether some constructs have a greater effect on the reasoning speed than others. Example 1 includes reasonably complex, although intentionally short, classes of lists and classifies on a fast laptop, using Protégé-OWL connected to FaCT++[13] or Pellet[14], in approximately 2 seconds. Running the fingerprint example through Pellet took between 80-170s to correctly classify various test proteins up to 450 elements long – other reasoners failed to return an answer. Because of the deeply nested 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.

Although not specifically designed for the biology domain, 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. OWL lists enable greater abstraction – "large and positive amino acid" - than their existing representation "Arginine or Lysine" and affords the possibility of using a reasoner to find those amino acids that could fit the looser specification that were not seen in the concrete collection of sequences from which the pattern was inferred. This is a potentially very powerful tool for investigating protein patterns. Additionally, classifying the patterns themselves, finding that one under-specified pattern subsumes another has intriguing biological possibilities.

5 Conclusion

In summary, we have described a design pattern for modeling sequences using 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. Algorithms based on deterministic finite automata as in standard regular expression matchers would almost certainly be faster. However, the two main advantages of this over traditional methods are; firstly, the expressivity of OWL can be used to model various levels of

[13] http://owl.man.ac.uk/factplusplus/

[14] http://www.mindswap.org/2003/pellet/

abstraction, which allows for underspecification (because we are often working with incomplete knowledge) at both the element (cell) level, and at the relative positional level for the entire structure; secondly, users can capture their knowledge in a single formalism and use standard DL reasoners to infer subsumption corresponding to pattern matching over classes of lists and to recognize lists of individuals as belonging to given classes.

Within OWL, there are certainly many alternative patterns to be investigated, such as directly linking elements together without intervening structure, and these are likely to share many of the same advantages as discussed in this paper. We have tried to demonstrate the usefulness of a general pattern – using protein sequences as an example application – and believe it can be applied to many different problems. In doing this we hope to avoid being distracted by the huge array of alternative models that could be built where the implementation is inextricable from the domain being modelled.

Critically, by allowing biologists to look at old problems in new ways, the OWL classification paradigm has allowed them to gain insights on the biology. 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.

Acknowledgements: This work was supported in part by the CO-ODE project funded by the Joint Information Services Committee (JISC) and by the HyOntUse Project (GR/S44686/1) funded by the UK Engineering and Physical Sciences Research Council (EPSRC). Our thanks go to Ulrike Sattler and Bijan Parsia for their advice.

6 References

1. Hirsh, H. and D. Kudenko. *Representing Sequences in Description Logics*. in *Fourteenth National Conference on Artificial Intelligence*. 1997.
2. Noy, N.F. and A. Rector, *N-ary relations*. 2004, Editors Draft, Semantic Web Best Practices Working Group, W3C.
3. Mulder, N.J., et al., *The InterPro Database, 2003 brings increased coverage and new features*. Nucleic Acids Res, 2003. **31**(1): p. 315-8.
4. Woodgett, J.R., K.L. Gould, and T. Hunter, *Substrate specificity of protein kinase C. Use of synthetic peptides corresponding to physiological sites as probes for substrate recognition requirements*. European Journal of Biochemistry, 1986. **161**(1): p. 177-184.
5. Sattler, U. *A Concept Language Extended with Different Kinds of Transitive Roles*. in *Deutsche Jahrestagung für Künstliche Intelligenz*. 1996: Springer Verlag.