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 modeled in ontologies. Yet the Web Ontology Language, OWL, contains no specific built in support for sequences or ordering. It does, however, have constructs that can be used to model many aspects of sequences, albeit imperfectly. This paper demonstrates two design patterns 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. Reasoning with standard DL reasoners works surprisingly well, but more efficient methods are almost certainly possible. The 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.

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, (which are made up of sequentially linked amino acids), carriages in a train, etc.
- Conceptually linked structures – *e.g.* documents, data structures, travel itinerary etc.

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 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. The OWL-S (OWL-Services) specification requires lists to describe the order of processes and provides an implementation of lists in OWL[2]

[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

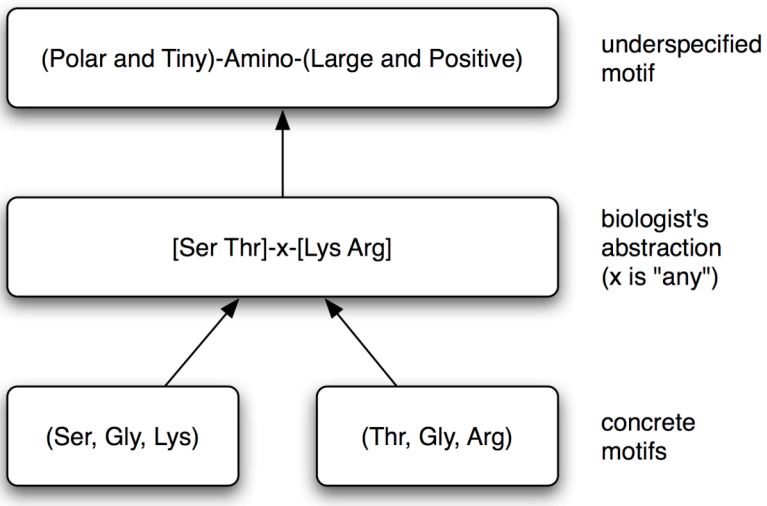

Figure 1: Levels of underspecification of motif pattern (ProteinKinaseC Phosphorylation Site)

however this is little more than the RDF vocabulary implemented in OWL likewise without further semantics.

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 and to infer that a sequence or class of sequences is subsumed by a particular class of lists – *i.e.* that it matches the pattern represented by that class of lists.

Sequences of amino acids (proteins) and nucleic acids (DNA or RNA) are fundamental to biology. We shall take proteins for our examples in this paper. Proteins are made up of sequences of linked amino acid residues. The twenty individual kinds of amino acids can be categorized along many different axes including size, polarity, charge, whether or not they contain sulphur, whether they are hydrophobic or hydrophilic (*i.e.* whether or not they are easily soluble in water), etc.

At a more abstract level, proteins can be thought of as containing "motifs", short sequences of amino acids that typically perform a particular function. Motifs are usually specified in terms of specific amino acids, but in many cases amino acids that share characteristics can be substituted for each other. Typically the specific motif sequences are extracted by special search engines, *e.g.* INTERPRO [5]. An example from biochemistry [8] is given in Figure 1. Concretely, a motif might consist of a

sequence of three specific amino acids, *e.g.* Serine, Glycine, Lysine. At a more abstract level, it might be thought of as being first a tiny polar amino acid, followed by any amino acid, terminated by a large positively charged amino acid. Numerous other short patterns of amino acid types are also examples of the same abstraction and might therefore share similar functions.

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 classes of sequences and then use the description logic 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".

A simple example ontology including all examples in this paper can be found on the web[3].

Patterns for Sequences

The idea of reasoning with sequential structures in OWL-DL using a tableaux reasoner may surprise some readers. However, we have found it surprisingly effective, and this has been investigated before. Hirsh & Kudenko [3] describe modeling strings as suffix trees expressed in a description logic to use subsumption to solve substring operations. However, their representation requires extensive rewriting. The relation of the resulting structures to the original lists is not immediately intuitive and, more importantly, the resulting structures grow as the square of the length of the list. As many of our lists run to several hundreds of elements, this represents a serious scaling problem.

We take a more intuitive approach related to that suggested by Hayes[4] and incorporated in the Semantic Web Best Practice Working Group's note on n-ary relations [6]. We review a range of constructs possible using this approach and their benefits in real world domains. By allowing them to look at old problems in new ways, the OWL classification paradigm has allowed users to gain insights on the biology.

In this paper, we propose two general design patterns for sequential entities:

- Lists considered as data structures, analogous to RDF lists.
- Sequences of linked entities, motivated by a more ontological view.

The first view is shown diagrammatically in Figure 2. We discuss the advantages and disadvantages of the two mechanisms plus possible variants and give practical examples from biology to demonstrate their utility.

[3] http://www.co-ode.org/ontologies/lists/
[4] Personal communication, Pat Hayes, 2004

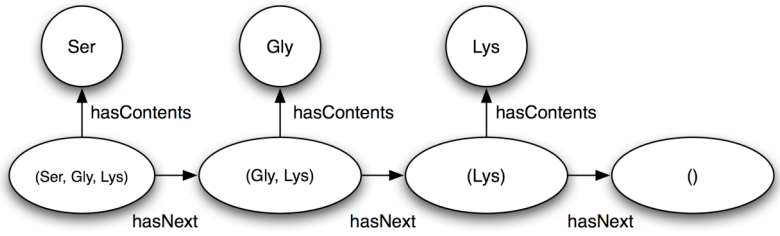

Figure 2: List data structure – simple example

```
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))
```

Figure 3: OWL vocabulary for lists as data structures in concrete abstract syntax

Modelling lists as data structures

In modeling 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 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, as already mentioned, we cannot use the RDF vocabulary in OWL-DL. Therefore, we must define a separate OWL vocabulary (Figure 3), an example of

which is shown diagrammatically in Figure 2. 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.

Simplified Syntax	OWL Abstract Syntax
A AND B	intersectionOf(A B)

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

Figure 4: Example of the definition of a class of OWL Lists of the form (Ser, Gly, Lys) in simplified syntax

A OR B	unionOf(A B)
NOT A	complementOf(A)
has_property SOME C	restriction(has_property someValuesFrom(C))
has_property ONLY C	restriction(has_property allValuesFrom(C))
has_property EXACTLY-n C	restriction(has_property cardinality(n, C)
B → A	subclassOf(B A)
A ←→ B	equivalentClass(A B)

Figure 5: Manchester Simplified Syntax

An example of a fully specified list is shown in Figure 4. In this and subsequent examples we use the simplified "Manchester Syntax"[5] used in the Protégé-OWL plugin[6]. The correspondence with OWL abstract syntax is given in Figure 5.

[5] http://www.co-ode.org/resources/reference/manchester_syntax/

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[7]). 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*. 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 the class of empty lists 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 an individual list, or class of lists, is empty.

Defining classes of lists

For simplicity we shall assume that the classes of potential contents of lists are disjoint. To take our example, the classes corresponding to types of amino acids are disjoint – *e.g.* no molecule can be both Serine and Lysine, or of any other two distinct amino acids. Furthermore, amino acids are themselves disjoint from other things in our model. Conveniently amino acids are conventionally abbreviated to either three letter or single letter codes; we will use the three letter codes for readability although the single letter codes are used in the examples on the Web for brevity.

Possible constructs are shown in Figure 6a along with a simplified syntax and examples. Sample class definitions illustrating the patterns are given in Figure 6b. We use a sugared shorthand syntax for lists that should be intuitive given the definitions and examples.

[6] http://protege.stanford.edu/overview/protege-owl.html
[7] http://www.w3.org/TR/owl-ref/#OWLDL

	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 6a. List simplified notation summary

	terminology	class definition using pattern
1	(A, B, C)	as per Fig 3
2	(A!)	List_only_As → List AND hasContents ONLY A AND isFollowedBy ONLY List_only_As
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 hasNext SOME ((List AND (hasContents SOME B)) OR List_AsFollowedByB) (variation including "AND hasContents SOME A" can also be added for "1 or more")
7	([A, B, C], B, C)	as per def 1 but substitute A for (A OR B OR C)
8	(hasProp some X, B, C)	as per def 1 but substitute A for (hasProp some X)
9	¬(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

Figure 6b: Example definitions for List Classes

Figure 7: Extract from inferred hierarchy of example lists

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 four significant differences:

- 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[8] 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 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. Note also the recursive definition required for this construct in line 6.
- 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.

  A small extract of the classified online example is shown in figure 7.

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

Limitations

There are a number of limitations on the expressive power of OWL lists:

- There is no way to define a class of "lists" so that it excludes 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[9]. In the absence of a full logical check, checking for cycles must be done separately. (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[10].
- 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`. 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`
  rather than
  `hasNext o hasNext o … o hasNext → isFollowedBy`
  OWL only includes the second, weaker, of these statements, because optimizations only exist currently for the simple implication [7] and not for the bi-implication[2].
- There is no way to define the class such that it subsumes just those classes mentioned in a class of lists, *e.g.* from the definition of the class of lists `L = (A, B, C)` to define a class `M` that subsumes precisely the classes `A`, `B`, and `C`. Although counter intuitive, this follows from the basic semantics of OWL. To say that the class `A` is a subclass of `M` would be to say that all individuals `a` in class `A` were members of `M` and hence of the list `L`. Clearly this is not generally the case. For example, all proteins contain amino acids bound in their chains, but not all amino acids are bound in proteins. The most practical way around this problem appears to be to use some level of meta reasoning to query the inferred knowledge base and return the information implicit in it. (Note that by contrast, if there is an inverse defined for the property `isFollowedBy`, say `isPrecededBy`, then it is possible to determine the class of members of an individual list made up of individuals, e.g.
  `List AND isPrecededBy theHeadOfMyListOfIndividuals`
- 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 Figure 6). 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.

[9] Personal communication, Ulrike Sattler, May 2006.
[10] and OWL 1.1

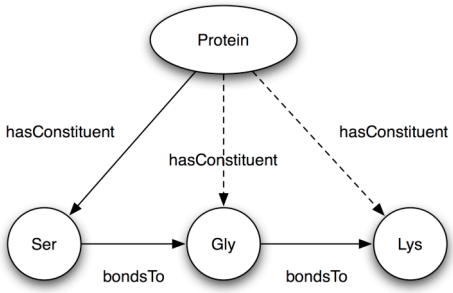

Figure 8: Protein as a whole constituted of a chain of amino acids

Modelling sequences directly as chains

Before returning to discuss practical experiences and the relation to regular expressions we first introduce a major alternative pattern.

Practically the above formulation should seem familiar to most programmers and software engineers. However, ontologists might raise questions. If an ontology is a representation of our conception of the world, what is represented by the notion of "List" in the previous section? Are proteins lists? Do Proteins contain lists? Consist of lists? At the class level, from a biological perspective, there is a notion of a Protein Primary Structure, which maps to this data structure. They contain amino acids (or to be pedantic, amino acid residues). However, at the level of an individual protein sequence, each amino acid residue is directly bound to the next without intervening structure.

An alternative, which deals with these concerns, is to represent the sequence directly – *e.g.* to say that a protein is a "chain" of amino acids, one bound to another, bound to another, etc. In effect we eliminate the notion of list altogether and replace the properties `hasNext` and `isFollowedBy` by physical interpretations, *e.g.* `bondsTo` and `bondsToInChain` as illustrated in Figure 8.

All of the above analysis holds except that it is unclear whether the notion of a chain of zero length is meaningful and, correspondingly, there is a question about how to terminate the chain. For termination in this case, we use simply a restriction on the final amino acid that it is not bound to anything further in the chain:

```
NOT (bindsToInChain SOME owl:Thing)
```

However, this modeling style still has problems. If we use the classifier, we will find that the protein chain "Serine bound to anything bound to Arginine bound to …" is a kind of Serine – the kind that is bound in this specific way, since it *is a* Serine with additional restrictions. This clearly does not fit with biologists' conceptualization of the world. Firstly, proteins are not considered kinds of amino acids, or even amino acid residues. In fact in most ontologies, the classes for proteins and amino acids would be disjoint. Secondly, proteins have many properties that do not pertain to amino acids – how they fold, their function, the gene that codes for them, etc. A protein is much more than just a chain of amino acids.

To capture our picture of a protein we need a representation of the protein itself as well as of the constituent chain of amino acids. It is the protein as a whole that is folded, has functions etc. This gives us the picture in Figure 8. All but the first `hasConstituent` link are dotted because they can potentially be inferred in OWL 1.1[11] [4] if we include the role inclusion axiom equivalent to:

```
hasConstituent o bondsTo → hasConstituent
```

Expressed in the proposed OWL 1.1 abstract syntax as:

```
SubPropertyOf(
        propertyChain(hasConstituent bondsTo) hasConstituent)
```

From an engineering standpoint it is clearly desirable to infer the constituent links. Otherwise, redundant information must be maintained independently. We can approximate these results using OWL1.0 by making `bondsTo` a subproperty of `hasConstituent` (which should be transitive), but this has the undesirable side effect of implying that each amino acid is a constituent of the previous one. Potentially, tools could temporarily filter out this nonsense result in the absence of adequate reasoning power.

A deeper discussion of the ontological issues with respect to data structures, entities in the world, constituents, etc. is far beyond the scope of this paper whose primary focus is engineering. For those concerned with upper ontologies, this representation fits with the multiplicative cognitive ontologies such as DOLCE [1], although here we use the inverse property `hasConstituent` rather than `constitutes`. The relationship between the sequence and the protein is the same as that between clay and statue. The Realist ontologies typified by BFO [2] would advocate alternative representations.

Discussion

On the one hand, expressing order in OWL models is a critical requirement for many users. On the other, many may find reasoning about lists using a tableaux reasoner surprising. There are at least five questions to ask:

- Is it useful?
- Does it produce the correct inferences?
- How expressive is it relative to the alternatives, in particular to regular expressions?
- Are there more efficient solutions computationally?
- Which pattern should be chosen for any particular application?

With respect to usefulness, the mechanisms described have been used with biologists to capture notions that they would otherwise find difficult to express. The basic scenario sketched in the introduction can be carried through in practice. 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 Serine

[11] http://www-db.research.bell-labs.com/user/pfps/owl/overview.html

or Threonine at this position. A regular expression is then made that captures this inference. What OWL lists enable is the abstraction – "large and positive". The abstraction is more expressive than the disjunction "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. Finally, classifying the patterns themselves, finding that one under-specified pattern subsumes another has intriguing biological possibilities.

With respect to subsumption, 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. As noted, OWL-DL is not sufficiently expressive to exclude all unintended constructs., particularly branching structures resulting from the incorrect use of `hasNext` and `isFollowedBy`. Additional constraints in the tools are required to avoid these cases if users are to be assured of the expected results.

Likewise, it is not possible to define classes of cyclical list-like structures. This is a major problem in some applications and but an advantage in others. For example, cyclical compounds play an important role in biology and are formed of true cycles of bound atoms. Being unable to represent cyclical compounds is a major limitation, although arguably not an issue of "lists". By contrast, 'life cycles' are not true cycles when considered from the point of view of the individual organism. When a parasite egg matures and goes through various transformations to reach sexual maturity and then produce another egg, it does not produce the same egg from which it originated, even though we typically draw such causal chains as cycles. The OWL representation correctly expresses our understanding of the world, which might be better described as a spiral than a cycle.

With regards to expressiveness, space does not allow an exhaustive proof, but Figure 6b indicates that the standard regular expression constructs can be represented, although at the cost of some rewriting. However, using OWL adds the ability to use abstract descriptions whose detailed structure can be inferred by the classifier, what biologists would usually call "motifs".

Practically, for the test-cases that have been run, computation is 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 machine, using Protégé-OWL connected to FaCT++[12] or Pellet[13], 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 large 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 – other reasoners failed to return an answer. 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. Note that the representation

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

proposed by Hirsh & Kudenko [3] require on the order of n^2 structures for a list of length n, making them impractical for sequences such as proteins that run to hundreds of elements or more.

As to the question of whether there are more computationally efficient ways of performing such reasoning, it would seem surprising if there were not. Algorithms based on deterministic finite automata as in standard regular expression matchers would almost certainly be faster. 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. One of the purposes of this paper is to provide motivation to extend such reasoners, and possibly the OWL specification.

Finally, as to which pattern to choose, this depends on both taste and ontological commitments. The list pattern corresponds much more closely to computer scientists' expectations and traditional data structures. If we interpret the `List` entities as reified relations, they provide a means of describing sequences of things that are not intrinsically linked, *e.g.* each leg of a multi-destination flight itinerary can exist on its own. It is only bound to the next flight by the steps in the passenger's itinerary. By contrast, the amino acids in a protein are physically bound one to the next to form a physical chain. Other molecules of the same amino acid exist that are not bound in the chain, but the individual amino acids in an individual protein molecule are physically bound, each to the next, in a way that the flights that make up an itinerary are not (as anyone who has ever missed a connection can testify).

Conclusion

In summary, we have described two closely related design patterns for describing sequences in OWL-DL. The first corresponds to the standard representation of linked lists as data structures; the second corresponds more closely to the ontological notion of sequences or chains of entities in the real world. We have used 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.

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. However, more efficient mechanisms are almost certainly possible.

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

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– *e.g.* in our biology example using protein sequences, to be able to work with notions such as "tiny, polar amino acid, followed by any amino acid, then by a large, positive amino acid" rather than "Serine or Threonine followed by anything, followed by Lysine or Arginine".

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.

References

1. Gangemi, A, Guarino, N, Masolo, C, Oltramari, A, and Schneider, L. Sweetening ontologies with DOLCE. in *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW02)*. 2002. Sigüenza, Spain.
2. Grenon, P, Smith, B, and Goldberg, L. Biodynamic Ontology: Applying BFO in the Biomedical Domain. in *Ontologies in Medicine*. 2004. Amsterdam: IOS Press.
3. Hirsh, H and Kudenko, D. Representing Sequences in Description Logics. in *Fourteenth National Conference on Artificial Intelligence*. 1997.
4. Horrocks, I and Sattler, U, Decidability of SHIQ with complex role inclusion axioms. *Artificial Intelligence*, 2004. 120 (1-2): p. 79-104.
5. Mulder, N J, et al., The InterPro Database, 2003 brings increased coverage and new features. *Nucleic Acids Res,* 2003. 31(1): p. 315-8.
6. Noy, N F and Rector, A, N-ary relations. 2004, Editors Draft, Semantic Web Best Practices Working Group, W3C.
7. Sattler, U. A Concept Language Extended with Different Kinds of Transitive Roles. in *Deutsche Jahrestagung für Künstliche Intelligenz*. 1996: Springer Verlag.
8. Woodgett, J R, Gould, K L, and Hunter, T, 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.