Owlifier: An Application for Creating Simple OWL Ontologies from Spreadsheet-Based Knowledge Descriptions

Shawn Bowers   Mark Schildhauer   Joshua Madin   Mathew Jones

Abstract

1 Introduction

2 Owlifier Syntax

An owlifier table defines an OWL-DL [?] ontology through a set of *blocks*. Each non-empty row in an owlifier table corresponds to a block. The type of the block is given in the first column of the row. The types of blocks supported by owlifier are as follows. It is assumed below that if any of the properties or concepts used in a block are defined, i.e., are not imported from another ontology, then they are added to the current ontology.

Import Block. Import blocks assign namespace labels to external ontologies. Each external ontology is imported into the current ontology. We refer to the ontologies of import blocks as *imported ontologies*. Using import blocks, concepts and properties of imported ontologies can be used within other blocks of the table. An import block has the form

`import`	*NS*	*URI*

where *NS* is a namespace label and *URI* is an OWL ontology URI. Concepts and properties from imported ontologies are referenced by prefixing the namespace label *NS* to the corresponding concept or property name in the normal way.

Concept Block. Concept blocks specify concept and subconcept relationships. A concept block has the form

`concept`	C_1	C_2	...	C_n

$(n \geq 1)$

where each concept C_i is asserted in the current ontology to subsume a concept C_{i+1}, for $1 \leq i < n$. Each C_i in a concept block induces a DL axiom

$$C_i \sqsubseteq C_{i+1}$$

If both C_i and C_{i+1} are imported concepts, we say that the block defines an "articulation" (i.e., mapping) between them.

Synonym Block. Synonym blocks define an equivalence relationship between concepts. A synonym block has the form

`synonym`	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where each concept C_i is asserted as being equivalent to concept C_{i+1} in the current ontology, for $1 \leq i < n$.

Overlap Block. Except in certain situations (described further below), defined concepts are assumed to be disjoint. Overlap blocks explicitly relax this assumption for a given set of concepts. An overlap block has the form

`overlap`	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where each concept C_i is allowed to share instances with each concept C_j, for $1 \leq i, j \leq n$. In particular, C_i and C_j are not defined to be disjoint concepts in the current ontology.

Property Block. Property blocks define the required *object* properties of concepts. A property block has the form

`property`	P	C_1	C_2	$\ldots$	C_n

$(n \geq 2)$

where P is an object property and each C_i is a concept, for $1 \leq i \leq n$. For every concept C_i, the property block induces the DL axiom

$$C_i \sqsubseteq \exists P.C_{i+1}$$

stating that each instance of C_i has, amongst possibly other things, a relationship through P to some instance of C_{i+1}. For example, the block

```
property hasPart Body Head Eye Retina
```

states that a body has at least one head, a head has at least one eye, and an eye has at least one retina.

Attribute Block. Attribute blocks are used to define the required *datatype* properties of concepts. An attribute block has the form

`attribute`	P	D	C_1	C_2	$\ldots$	C_n

$(n \geq 1)$

where P is a datatype property, each C_i is a concept for $1 \leq i \leq n$, and D is a datatype (`anyValueType`, `string`, `int`, etc.). For every concept C_i, the property block induces the DL axiom

$$C_i \sqsubseteq (\exists P.D) \sqcap (\leq 1\ P)$$

stating that each instance of C_i has, amongst possibly other things, a relationship through P to exactly one data value of type D.

Value Block. Value blocks define required datatype property *constant values* for concepts. A value block has the form

`value`	P	V	C_1	C_2	$\ldots$	C_n

$(n \geq 1)$

where P is a datatype property, C_i is a concept for $1 \leq i \leq n$, and V is a datatype value. For each concept C_i, the value block induces the DL axiom

$$C_i \sqsubseteq (V \in P)$$

stating that each instance of C_i has a value V for property P. The value restrictions stated by value blocks are often used for defining so-called *value partitions* [**?**].

Inverse Block. Inverse blocks state that two object properties are inverses of each other. That is, for inverse properties P_1 and P_2 and concept instances O_1 and O_2, if $P_1(O_1) = O_2$, then $P_2(O_2) = O_1$. An inverse block has the form

`inverse`	P_1	P_2

where P_1 and P_2 are object properties.

Transitive Block. Transitive blocks state that a property is transitive. That is, if P is transitive and a concept instance O_1 is related to an instance O_2 by P, and O_2 is related to an instance O_3 by P, then O_1 is also by definition related to O_3 by P. A transitive block has the form

`transitive`	P

where P is an object property.

Cardinality Block. Cardinality blocks state the maximum number of properties P an instance of a concept may have. Cardinality blocks have the form

`cardinality`	N	P	C_1	C_2

where N is the maximum number of properties P that instances of concept C_1 may have to instances of concept C_2. A cardinality block induces the DL axiom

$$C_1 \sqsubseteq (\leq N P.C_2)$$

stating that each instance of C_1 can be related to at most N unique instances of C_2 via P. For example, the blocks

```
cardinality 1 hasPart Body Head
cardinality 2 hasPart Head Eye
```

states that a body has at most one head and a head has at most two eyes.

Sufficient Block. Sufficient blocks state that any instance having a property P to an instance of a concept C_2 is a sufficient condition for being an instance of a concept C_1. A sufficient block has the form

`sufficient`	P	C_1	C_2

where C_1 is the target concept (i.e., denoting the concept definition), P is the sufficient property, and C_2 is the sufficient concept. A sufficient block induces the DL axiom

$$C_1 \equiv \exists P.C_2$$

Sufficient blocks provide a mechansism to construct simple class definitions (i.e., classes defined precisely by other classes), primarily for use with value partitions.

Description Block. Description blocks assign plain-text definitions to concepts and properties. A description block has the form

`description`	T	S

where T is either a property or a concept and S is a description string.

Note Block. Note blocks add comments to the current ontology, and are ignored by owlifier. A note block has the form

`note`	S

where S is a comment string.

*** Say something about relaxing block syntax ... to make it easier to specify ontologies. Also, allow blocks to be given in any order.

3 Owlifier Reasoning

e.g., Disjoint Concept Inference. Need to describe here when we say two concepts are disjoint. Other inferences are now possible as well.

Errors:

- Blocks with syntactic errors
- Inverse properties can be between at most two properties. For instance, inverse(P1,P2) and inverse(P1,P3) is not allowed.
- At most one description is allowed per property or concept.
- Property and concept names must be disjoint

Warnings:

- Cyclic concept hierarchies
- Re-definition of imported concepts (have to define what this means)
- Introduction of an inconsistency (can we show that this will never happen in a fully defined ontology, i.e., one without imports)?
- ...

4 Owlifier Examples

- Simple example, no imports, no warnings
- Extension example
- Articulation example

4.1 Owlifier for OBOE

How it works with OBOE.

5 Owlifier API

Flags:

- Turn on/off consistency checking/validation
- Output format (OWL/RDF,
- Output inferred axioms
- Ontology URI to use

6 Discussion

Implementation, etc.