

Extending OWL2 Manchester Syntax to Include Missing Features from OWL2 Abstract Syntax

Björn Gehrke¹, Till Mossakowski¹

¹Otto von Guericke University Magdeburg, Universitätsplatz 2, 39106 Magdeburg, Germany

Abstract

There are multiple serialisations available to store OWL2 ontologies. While the language structure is specified in Abstract Syntax, a more human-friendly serialisation is Manchester Syntax. As both are serialisations of OWL2, they should be equipotent. However, some features are known to be inexpressible in Manchester Syntax. Most commonly known is the lack of expressing general concept inclusions in Manchester Syntax. This paper analyses the Manchester Syntax grammar and identifies a total of 19 inexpressible features and eight minor errors in the current Manchester Syntax grammar. Furthermore, changes to the Manchester Syntax grammar are proposed to make it equipotent to Abstract Syntax. The proposed changes are implemented in the OWL API¹.

Keywords

OWL2, Manchester Syntax, GCI


1. Introduction

The “Web Ontology Language 2” (OWL2) [1] is commonly used for ontology development. There are multiple syntax styles to serialise an OWL2 document. The language structure is specified in Functional Syntax (or Abstract Syntax). The language’s capabilities, features, and limits are defined with it. Hence, every serialisation should be as expressive as Abstract Syntax and translations between all serialisations should be possible.


Another syntax is called Manchester Syntax. It is created specifically to be more human-readable and user-friendly. This is achieved by grouping the content in so-called frames around single entities instead of storing plain axioms, as other styles do. This leads to a more natural way of reading and writing axioms. However, compared to Abstract Syntax, certain statements cannot be expressed in Manchester Syntax. Those include, among others, general concept inclusions (GCI). [2]


In the context of OWL2, many axioms refer to a named class set in relation to classes. Those classes can be named themselves or anonymous. Anonymous classes express a set of individuals using complex class expressions without explicitly naming them.

¹<https://github.com/owlcs/owlapi>

 DL 2023: 36th International Workshop on Description Logics, September 2–4, 2023, Rhodes, Greece

 bjoern.gehrke@ovgu.de (B. Gehrke); till.mossakowski@ovgu.de (T. Mossakowski)

 © 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

For example, consider the following:

```
SubClassOf ( ObjectIntersectionOf( :a :b ) :c )
```

This expression cannot be expressed in Manchester Syntax directly as the class frame only allows a named class and no complex class expression.

Commonly used editors like Protégé¹ allow adding such General Class Axioms in a Manchester-like style. The previous example would be written as

```
:a and :b SubClassOf :c
```

This serialisation, however, is not valid Manchester Syntax and trying to save ontologies containing such axioms in Manchester Syntax results in those axioms being dropped.

2. Translation between Manchester Syntax and Abstract Syntax

The W3C note for OWL2 Manchester Syntax contains an algorithm to translate Manchester Syntax to Abstract Syntax. This translation is achieved by first converting every frame to multiple simple frames with the same subject. Therefore, previously grouped axioms relating to the same entity exist as single axioms. Now each frame expresses a single axiom which can be transformed using a transformation function T_{MS} . T_{MS} is given by a table E_{MS} recursively defining the transformation for each possible input. T_{MS}^{-1} is, therefore, the inverse of the function working as a reverse lookup in E_{MS} . The translation is depicted in Figure 1. [3]

It is stated to run the algorithm in reverse to convert an ontology from Abstract Syntax to Manchester Syntax. However, the steps provided in the algorithm - especially the transformation function T_{MS} - are written from a Manchester Syntax point of view, making T_{MS} an injective function from Manchester Syntax to Abstract Syntax. T_{MS} cannot be bijective as at least general concept inclusions are known to be not supported in Manchester Syntax.

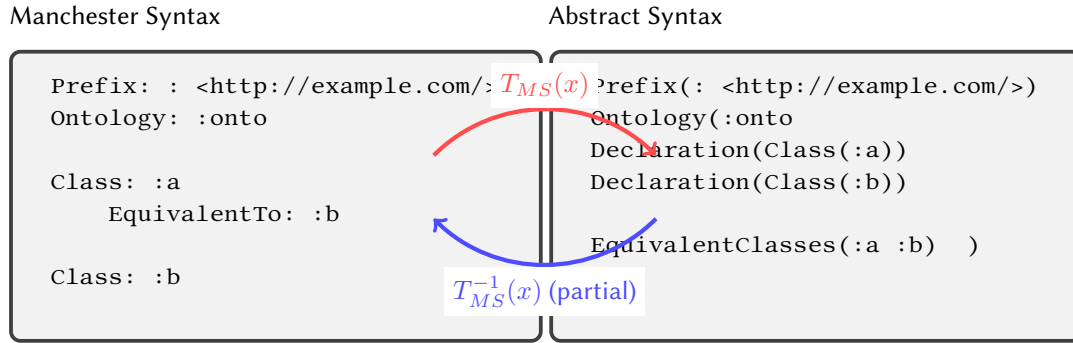
3. Missing features

The existing transformation table E_{MS} does not map axioms directly but uses a meta-language describing how a given ontology in Manchester Syntax can be transformed to Abstract Syntax. Keeping this style, for each rule r_i in the Abstract Syntax grammar containing terminal elements, there must be an entry e in E_{AS} such that $T_{AS}(e) = r$. Therefore, each non-terminal element v_i in e is replaced by $T_{AS}(v_i)$.

The existing translation specifications utilise T_{MS}^{-1} for translation from Abstract Syntax to Manchester Syntax, although T_{MS} is not bijective. To find missing features in Manchester Syntax, T_{MS}^{-1} is redefined as a new partial transformation function T_{AS} transforming Abstract Syntax to Manchester Syntax. It is defined by a transformation table E_{AS} . Using the whole grammar [4], every single rule r (producing an axiom, expression, and other features) is mapped to their Manchester Syntax representation $T_{MS}^{-1}(r)$, if possible. Otherwise, it is added to a set

¹<https://protege.stanford.edu/>

Figure 1: Translation from Manchester Syntax to Abstract Syntax and vice versa use a transformation function T_{MS} . T_{MS} is defined for each construct in Manchester Syntax. To translate Abstract Syntax to Manchester Syntax the standard states to use the reverse function T_{MS}^{-1} .



M . At the end, $M = \{ x \mid \nexists y : T_{AS}(x) = T_{MS}^{-1}(x) = y \}$ contains all inexpressible features of Manchester Syntax.

In total, there are 27 features found not expressible in Manchester Syntax when strictly following the existing grammar and transformation function. Those include spelling mistakes such as “FunctionalObjectProperty” instead of “ObjectFunctionalProperty” and other minor errors in E_{MS} such as annotations being required in classFrame rule. Excluding such errors, 19 features are left that are truly inexpressible. Table 1 lists the 19 features not supported in Manchester Syntax.

The lack of expressivity of Manchester Syntax can be remedied a bit using conservative extensions. Namely, an ontology in Abstract Syntax can be extended conservatively, and then this conservative extension (which is even purely definitional) can be mapped to an equivalent ontology in Manchester Syntax. This is possible if the Abstract Syntax ontology does not use the identified missing “DataSomeValuesFrom”, “DataAllValuesFrom”, or “Declarations” features and works as follows:

A general concept inclusion $\{C \sqsubseteq A\}$ can be written as $\{B \equiv C, B \sqsubseteq A\}$ with B being a new atomic concept, C a complex concept and A a complex or atomic concept.

In the same manner, a complex role inclusions $\{t \sqsubseteq r\}$ can be written as $\{s \equiv t, s \sqsubseteq r\}$ and any other missing feature involving complex object property expressions e.g. $\{\text{SymmetricObjectProperty}(t)\}$ as $\{s \equiv t, \text{SymmetricObjectProperty}(s)\}$ with s being an atomic role, t a complex role, and r a complex or atomic role.

However, working with such conservative extensions often is inconvenient, because the ontology will be cluttered with additional (often rather artificial) extra concepts and roles. Therefore, we will extend Manchester Syntax in a way that directly supports all Abstract Syntax features.

4. Grammar and translation table extensions

Having identified inexpressible features of OWL2 Manchester Syntax, the following proposes changes and additions to the existing Manchester Syntax grammar and translation table E_{MS} .

Table 1

Inexpressible features in Manchester Syntax according to the translation table E_{MS} . The table shows features in Abstract Syntax.

Feature in Abstract Syntax
DataSomeValuesFrom ($dp_1 \dots dp_n dr$) $\forall n > 1$
DataAllValuesFrom ($dp_1 \dots dp_n dr$) $\forall n > 1$
Declaration (<i>annos entity</i>)
SubObjectPropertyOf (<i>annos</i> ObjectInverseOf (<i>subOp</i>) <i>superOpExpr</i>)
SubObjectPropertyOf (<i>annos</i> ObjectPropertyChain (<i>opEx₁</i> ... <i>opEx_n</i>) ObjectInverseOf (<i>superOp</i>))
ObjectPropertyDomain (<i>annos</i> ObjectInverseOf (<i>op</i>) <i>desc</i>)
ObjectPropertyRange (<i>annos</i> ObjectInverseOf (<i>op</i>) <i>desc</i>)
InverseObjectProperties (<i>annos</i> ObjectInverseOf (<i>op</i>) <i>opExpr</i>)
FunctionalObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
InverseFunctionalObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
ReflexiveObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
IrreflexiveObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
SymmetricObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
AsymmetricObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
TransitiveObjectProperty (<i>annos</i> ObjectInverseOf (<i>op</i>))
ObjectPropertyAssertion (<i>annos</i> ObjectInverseOf (<i>op</i>) <i>source target</i>)
NegativeObjectPropertyAssertion (<i>annos</i> ObjectInverseOf (<i>op</i>) <i>source target</i>)
SubClassOf (<i>annos subClassExpression superClassExpression</i>)
HasKey (<i>annos desc</i> (<i>opExpr₁</i> ... <i>opExpr_n</i>) ($dp_1 \dots dp_m$)) $\forall n \geq 0, m \geq 0, n + m > 0$

It adds support for all missing features. All changes are analysed for backwards compatibility.

The proposed grammars utilise the meta-rules described in [3, Section 2]. Additionally, minor errors (like spelling mistakes) are corrected implicitly.

DataSomeValuesFrom and DataAllValuesFrom Although practically not usable, as data ranges with arity greater than one are currently not expressible in Abstract Syntax either[4, Section 7], adding the capability to Manchester Syntax removes the need to remember and adjust this once expressing data ranges with arity greater than one is possible. This feature can be added by allowing tuples of class expressions and single class expressions in the rule for restriction. The tuples must be surrounded by parenthesis to ensure the grammar stays unambiguous. Consequently, the following entries and rules must be added to E_{MS} and the grammar, respectively.

Manchester Syntax	Abstract Syntax
($desc_1, \dots, desc_n$) some dr	DataSomeValuesFrom($T(desc_1) \dots T(desc_n) T(dr)$)
($desc_1, \dots, desc_n$) only dr	DataAllValuesFrom($T(desc_1) \dots T(desc_n) T(dr)$)

restriction ::= ...

- | ' (' **dataPropertyExpressionList** ') ' 'some' **dataPrimary**
- | ' (' **dataPropertyExpressionList** ') ' 'only' **dataPrimary**

Annotations in declarations Allowing declarations to be annotated can be achieved in multiple ways. One way is to change the semantics of an `Annotations: frame` section inside a frame. According to T_{MS} , this is currently mapped to an `AnnotationAssertion` axiom in Abstract Syntax with the frames subject as annotation subject. Instead, this annotation could be added to the `Declaration` axiom.

A second approach is to allow the `Annotations: section - annotations` non-terminal in the Manchester Syntax grammar - in front of the IRI of a frame's subject. In contrast to the first proposal, this change keeps backward compatibility as annotations are optional. Consequently, the following entries and rules must be changed in E_{MS} and the grammar, respectively.

Frame	Declaration
AnnotationProperty: <i>annotations IRI ...</i>	Declaration ($T(\textit{annotations})$ AnnotationProperty (<i>IRI</i>))
Class: <i>annotations IRI ...</i>	Class ($T(\textit{annotations})$ Class (<i>IRI</i>))
DataProperty: <i>annotations IRI ...</i>	Declaration ($T(\textit{annotations})$ DataProperty (<i>IRI</i>))
Datatype: <i>annotations IRI ...</i>	Declaration ($T(\textit{annotations})$ Datatype (<i>IRI</i>))
Individual: <i>annotations IRI ...</i>	Declaration ($T(\textit{annotations})$ NamedIndividual (<i>IRI</i>))
ObjectProperty: <i>annotations IRI ...</i>	Declaration ($T(\textit{annotations})$ ObjectProperty (<i>IRI</i>))

```

datatypeFrame ::= 'Datatype:' [ annotations ] Datatype ...
dataPropertyFrame ::= 'DataProperty:' [ annotations ] ↔
    dataPropertyIRI ...
annotationPropertyFrame ::= 'AnnotationProperty:' ↔
    [ annotations ] annotationPropertyIRI ...
individualFrame ::= 'Individual:' [ annotations ] individual ...

```

Features with object property expressions as subject Many features inexpressible in Manchester Syntax have object property expressions as a subject. In Manchester Syntax, however, the subject of a frame can only be an IRI, thus an object property. Changing the subject from IRI to object property expression adds support for the following axioms to be used with object property expressions:

- SubObjectPropertyOf
- ObjectInverseOf
- ObjectPropertyDomain
- ObjectPropertyRange
- InverseObjectProperties
- FunctionalObjectProperty
- InverseFunctionalObjectProperty

- ReflexiveObjectProperty
- IrreflexiveObjectProperty
- SymmetricObjectProperty
- AsymmetricObjectProperty
- TransitiveObjectProperty
- ObjectPropertyAssertion
- NegativeObjectPropertyAssertion

Nevertheless, simply changing the ObjectProperty: frames subject would also allow statements describing AnnotationAssertion axioms. However, AnnotationAssertion axioms can only have an IRI as the subject. Hence, the grammar must ensure such cases are not possible. Apart from the AnnotationAssertion axiom, all derivations of the objectPropertyFrame non-terminal produce valid axioms with the proposed changes. However, complex object property expressions cannot be declared. Hence, no additional entry in the declarations table is added. In addition to the changes in “Annotations in declarations”, leading annotations are not allowed if a complex object property expression is used. Consequently, the following entries and rules must be added to or changed in *EMS* and the grammar, respectively. The following table only contains the translation for ObjectPropertyDomain. All other axioms mentioned above are translated similarly. Note that the non-terminal OPE is only introduced for better readability.

Manchester Syntax	Abstract Syntax
ObjectProperty: <i>opExpr</i> Domain: <i>annos</i> description	ObjectPropertyDomain(T(annotations) <i>T(opExpr)</i> T(description))

OPE ::= objectPropertyExpression

simpleObjectPropertySection ::= complexObjectPropertySection
| 'Annotations:' **annotationAnnotatedList**

complexObjectPropertySection ::=
 'Domain:' **descriptionAnnotatedList**
| 'Range:' **descriptionAnnotatedList**
| 'Characteristics:' \leftrightarrow
 objectPropertyCharacteristicAnnotatedList
| 'SubPropertyOf:' \leftrightarrow
 objectPropertyExpressionAnnotatedList
| 'EquivalentTo:' \leftrightarrow
 objectPropertyExpressionAnnotatedList
| 'DisjointWith:' \leftrightarrow
 objectPropertyExpressionAnnotatedList

```

| 'InverseOf:'          ↔
    objectPropertyExpressionAnnotatedList
| 'SubPropertyChain:'  ↔
    annotations OPE 'o' OPE { 'o' OPE }

objectPropertyFrame ::=
    'ObjectProperty:' [ annotations ] IRI ↔
        { simpleObjectPropertySection }
| 'ObjectProperty:' OPE { complexObjectPropertySection }

```

General concept inclusions and HasKey A naive approach extends the misc frame to include GCIs and HasKey axioms. It already contains all axioms not fitting to any other frame. General concept inclusions containing complex class expressions instead of a simple class IRI could be considered as such. Similarly, the HasKey axiom can be added there too. However, this approach introduces two new keywords breaking backwards-compatibility.

A much more natural way is to extend the class frame. Currently, it is possible to express a SubClassOf axiom only with a simple IRI as a subclass. Changing this IRI to description - a complex class expression - allows expressing general concept inclusions. Furthermore, this change also allows expressing HasKey axioms with complex class expressions. Additionally, an IRI being a valid class expression ensures the backwards compatibility of this change.

Similar to changing the object property frames subject, changing the class frames subject might lead to invalid axioms. In addition, the DisjointWith section must also be excluded. It is mapped to the DisjointUnionOf axiom in Abstract Syntax. Similar to the AnnotationAssertion axiom, the DisjointUnionOf axiom allows only a class and not a complex class expression [4]. Consequently, the following entries and rules must be changed in or added to E_{MS} and the grammar, respectively.

Manchester Syntax	Abstract Syntax
Class: <i>subDesc</i> SubClassOf: <i>annos superDesc</i>	SubClassOf($T(annos) T(subDesc) T(superDesc)$)
Class: <i>desc₁</i> EquivalentTo: <i>annos desc₂</i>	EquivalentClasses($T(annos) T(desc_1) T(desc_2)$)
Class: <i>desc₁</i> DisjointWith: <i>annos desc₂ ... desc_n</i>	DisjointClasses($T(annos) T(desc_1) ... T(desc_n)$)
Class: <i>desc</i> HasKey: <i>annos properties</i>	HasKey($T(annos) T(desc) T(properties)$)

```

simpleClassSection ::= complexClassSection
| 'Annotations:'      annotationAnnotatedList
| 'DisjointUnionOf:' [ annotations ] description2List

```

```

complexClassSection ::= 'SubClassOf:' descriptionAnnotatedList
  | 'EquivalentTo:' descriptionAnnotatedList
  | 'DisjointWith:' descriptionAnnotatedList
  | 'HasKey:' [ annotations ]
    ( objectPropertyExpression | dataPropertyExpression )
    { objectPropertyExpression | dataPropertyExpression }

classFrame ::=
  'Class:' [ annotations ] IRI { simpleClassSection }
  | 'Class:' description { complexClassSection }

```

Figure 2: Set of axioms that can be expressed with the proposed changes. Highlighted in red are constructs that previously were not expressible and, therefore, not allowed in Manchester Syntax. Note that it is generally impossible to declare a datatype with an arity of more than one in OWL2 DL.

```

ObjectProperty: Annotations: rdfs:comment "comment" op1
ObjectProperty: inverse op1
  Characteristics: Functional, Symmetric

DataProperty: dp1      DataProperty: dp2
Datatype: dt1          # fictional datatype with arity of 2

Class: a      Class: b      Class: c
Class: a and b
  SubClassOf: (dp1, dp2) some dt1

```

5. Implementation

We implemented the presented extensions in the reference implementation OWL API in the pull requests 1050² for version 5 and 1104³ for version 4. They are available in the OWL API version 5.5.0 and waiting to be merged for version 4.

During the implementation, it turned out, that the OWL API already supports object property expressions as frame subjects: Their implementation follows the proposed changes of section 4. Furthermore, and as mentioned, the OWL API does not support `DataSomeValueFrom` and `DataAllValuesFrom` for data ranges with an arity greater than one. Therefore, changes in the underlying data types are required. This is a major change with no current practical use. Hence, these two features are not implemented in the OWL API.

²<https://github.com/owlcs/owlapi/pull/1050>

³<https://github.com/owlcs/owlapi/pull/1104>

6. Conclusion

A translation from OWL2 Abstract Syntax to OWL2 Manchester Syntax was constructed yielding 19 inexpressible features (27 including minor errors in E_{MS}) in Manchester Syntax. A change of the existing grammar and translation table is proposed for each such inexpressible feature. With these changes, all constructs expressible in OWL2 DL are expressible in Manchester Syntax, marking both syntaxes equipotent. A collection of axioms in Figure 2 highlight the features and show what constructs can now be expressed in Manchester Syntax.

The proposed changes have been implemented in the OWL API and are available from version 5.5.0 and once merged will be available in the next version 4 release. With these changes, the more human-readable Manchester Syntax can be used for serious ontology development without drawbacks. While many developers use tools such as Protégé independent of the serialisation, these features are still essential: For example, manually resolving conflicts or tracking changes when developing with a version control system such as git is much simpler in Manchester Syntax than in Abstract Syntax or XML.

References

- [1] W3C-Owl-Working-Group, OWL 2 Web Ontology Language Document Overview (Second Edition), Recommendation, W3C, 2012. <https://www.w3.org/TR/2012/REC-owl2-overview-20121211/>. Latest version available at <https://www.w3.org/TR/owl2-overview/>.
- [2] R. S. Uli Sattler, Being complex on the left-hand-side: General concept inclusions, Ontogenesis (2012).
- [3] M. Horridge, P. F. Patel-Schneider, Owl 2 web ontology language manchester syntax, W3C Working Group Note (2012).
- [4] B. Motik, P. Patel-Schneider, B. Parsia, OWL 2 Web Ontology Language Structural Specification and Functional-Style Syntax (Second Edition), Recommendation, W3C, 2012. <https://www.w3.org/TR/2012/REC-owl2-syntax-20121211/>. Latest version available at <https://www.w3.org/TR/owl2-syntax/>.