

---

# DAML+OIL Technical Detail

Ian Horrocks

`horrocks@cs.man.ac.uk`

University of Manchester

Manchester, UK

# Talk Outline

---

## Overview of language design and motivation

### Basic features

- ☞ quick review of walkthru

### Advanced features

- ☞ details not (sufficiently) covered in the walkthru

### Tricks of the Trade

- ☞ getting the most out of DAML+OIL

### Limitations

- ☞ what it can't do

### Implementation challenges

---

# Overview of Language Design and Motivation

# DAML+OIL: a Semantic Web Ontology Language

---

- ☞ Most existing Web resources only human understandable
  - Markup (HTML) provides **rendering information**
  - Textual/graphical information for **human consumption**
- ☞ Semantic Web aims at **machine understandability**
  - **Semantic** markup will be added to web resources
  - Markup will use **Ontologies** for shared understanding
- ☞ Requirement for a suitable ontology language
  - Compatible with existing Web standards (XML, RDF)
  - Captures common KR idioms
  - Formally specified and of adequate expressive power
  - Amenable to machine processing
    - ➔ Can provide reasoning support
- ☞ DAML+OIL language developed to meet these requirements

# DAML+OIL Language Overview

---

DAML+OIL is an **ontology** language

- ☞ Describes **structure** of the domain (i.e., a Tbox)
  - RDF used to describe specific **instances** (i.e., an Abox)
- ☞ Structure described in terms of **classes** and **properties**
- ☞ Ontology consists of set of **axioms**
  - E.g., asserting class subsumption/equivalence
- ☞ Classes can be names or **expressions**
  - Various **constructors** provided for building class expressions
- ☞ **Expressive power** determined by
  - Kinds of class (and property) constructor supported
  - Kinds of axiom supported

---

# Basic Features

# Classes and Axioms

---

Ontology consists of set of **axioms**, e.g., asserting facts about **classes**:

```
<daml:Class rdf:ID="Animal" />
```

```
<daml:Class rdf:ID="Man">  
  <rdfs:subClassOf rdf:resource="#Person" />  
  <rdfs:subClassOf rdf:resource="#Male" />  
</daml:Class>
```

```
<daml:Class rdf:ID="MarriedPerson">  
  <daml:intersectionOf rdf:parseType="daml:collection">  
    <daml:Class rdf:about="#Person" />  
    <daml:Restriction daml:cardinality="1">  
      <daml:onProperty rdf:resource="#hasSpouse" />  
    </daml:Restriction>  
  </daml:intersectionOf>  
</daml:Class>
```

# Properties

---

Can also assert facts about properties, e.g.:

```
<daml:ObjectProperty rdf:ID="hasParent" />
```

```
<daml:UniqueProperty rdf:ID="hasMother">  
  <rdfs:subPropertyOf rdf:resource="#hasParent" />  
  <rdfs:range rdf:resource="#Female" />  
</daml:UniqueProperty>
```

```
<daml:TransitiveProperty rdf:ID="descendant" />
```

```
<daml:ObjectProperty rdf:ID="hasChild">  
  <daml:inverseOf rdf:resource="#hasParent" />  
</daml:ObjectProperty>
```

```
<daml:ObjectProperty rdf:ID="hasMom">  
  <daml:samePropertyAs rdf:resource="#hasMother" />  
</daml:ObjectProperty>
```



# Datatypes

---

Can use XMLS datatypes and values instead of classes and individuals:

```
<daml:DatatypeProperty rdf:ID="age">
  <rdf:type rdf:resource=".../daml+oil#UniqueProperty"/>
  <rdfs:range rdf:resource=".../XMLSchema#nonNegativeInteger"/>
</daml:DatatypeProperty>
```

```
<xsd:simpleType name="over17">
  <xsd:restriction base="xsd:positiveInteger">
    <xsd:minInclusive value="18"/>
  </xsd:restriction>
</xsd:simpleType>
```

```
<daml:Class rdf:ID="Adult">
  <daml:Restriction>
    <daml:onProperty rdf:resource="#age"/>
    <daml:hasClass rdf:resource="...#over17"/>
  </daml:Restriction>
</daml:Class>
```

# Individuals

---

Can also assert facts about individuals, e.g.:

```
<Person rdf:ID="John" />
```

```
<Person rdf:ID="Mary" />
```

```
<rdf:Description rdf:about="#John">
```

```
  <hasParent:resource="#Mary" />
```

```
  <age>25</age>
```

```
</rdf:Description>
```

```
<rdf:Description rdf:about="#John">
```

```
  <differentIndividualFrom:resource="#Mary" />
```

```
</rdf:Description>
```

```
<rdf:Description rdf:about="#Clinton">
```

```
  <sameIndividualAs:resource="#BillClinton" />
```

```
</rdf:Description>
```

---

# Advanced Features

# Overview of Class Expressions

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \dots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \dots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$ Male
oneOf	$\{x_1 \dots x_n\}$	{john, mary}
toClass	$\forall P.C$	$\forall$ hasChild.Doctor
hasClass	$\exists P.C$	$\exists$ hasChild.Lawyer
hasValue	$\exists P.\{x\}$	$\exists$ citizenOf.{USA}
minCardinalityQ	$\geq_n P.C$	$\geq 2$ hasChild.Lawyer
maxCardinalityQ	$\leq_n P.C$	$\leq 1$ hasChild.Male
cardinalityQ	$=_n P.C$	$= 1$ hasParent.Female

👉 XMLS **datatypes** can be used in restrictions

👉 Arbitrary **nesting** of constructors

- E.g.,  $\forall$ hasChild.(Doctor  $\sqcup$   $\exists$ hasChild.Doctor)

# Class Names

---

Most basic components of class expressions are **names**

- 👉 E.g., Person, Building
- 👉 Two **built-in** (pre-defined) class names:
  - **Thing** — class whose extension is whole (object) domain
  - **Nothing** — class whose extension is empty
- 👉 They are just “syntactic sugar”
  - $\text{Thing} \equiv C \sqcup \neg C$  for any class  $C$
  - $\text{Nothing} \equiv \neg \text{Thing}$

# Class Expressions: Restrictions

---

➡ Restrictions are classes: class of all objects satisfying restriction

➡ Basic structure is **property** plus restrictions on

- **type** and/or
- **number**

of objects that can be related to members of class via that property

# toClass Restrictions

---

➡ E.g.:

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasParent" />  
  <daml:toClass rdf:resource="#Person" />  
</daml:Restriction>
```

class of objects all of whose parents are persons

➡ Analogous universal quantification ( $\forall$ ) in FOL

➡ Analogous to box ( $\square$ ) in modal logic

# toClass Restrictions

---

- ➡ Can be seen as local/relativised property **range**

```
<daml:Class rdf:about="#Person">
  <rdfs:subClassOf>
    <daml:Restriction>
      <daml:onProperty rdf:resource="#hasParent"/>
      <daml:toClass rdf:resource="#Person"/>
    </daml:Restriction>
  </rdfs:subClassOf>
</daml:Class>
```

- ➡ Conversely, range is like asserting toClass restriction w.r.t. Thing

- ➡ Some “strange” inferences:

- instances with no conflicting property assertions may not be members of class (open world) — c.f. peter
- instances (provably) without any such property are members of class — c.f. paul



# hasClass Restrictions

---

➡ E.g.:

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:hasClass rdf:resource="#Republican" />  
</daml:Restriction>
```

class of objects that have some friend that is a Republican

➡ Analogous existential quantification ( $\exists$ ) in FOL

➡ Analogous to diamond ( $\diamond$ ) in modal logic

➡ Individuals with no relevant property assertions may still be members of class (incomplete knowledge)

# hasValue Restrictions

---

👉 E.g.:

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:hasValue rdf:resource="#Nixon" />  
</daml:Restriction>
```

class of objects that have some friend that is Nixon

👉 Just a special case of hasClass using oneOf

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:hasClass>  
    <daml:oneOf rdf:parseType="daml:collection">  
      <rdf:Description rdf:about="#Nixon">  
    </daml:oneOf>  
  </daml:hasClass>  
</daml:Restriction>
```

# cardinality Restrictions

---

👉 E.g.:

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:minCardinalityQ>2</daml:minCardinalityQ>  
  <daml:hasClassQ rdf:resource="#Republican" />  
</daml:Restriction>
```

class of objects that have at least 2 friends that are Republicans

👉 Can specify min, max and exact cardinalities

- exact is shorthand for max plus min pair

👉 minCardinalityQ is generalisation of hasClass, e.g.:

```
<daml:Restriction daml:minCardinalityQ=1>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:hasClassQ rdf:resource="#Republican" />  
</daml:Restriction>
```

equivalent to hasClass Republican.

# cardinality Restrictions

---

- ➡ Also exist versions without qualifying concepts, e.g.:

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:minCardinality>3</daml:minCardinality>  
</daml:Restriction>
```

class of objects that have at least 3 friends

- ➡ Same as Q version with qualifying class as Thing

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:minCardinalityQ>3</daml:minCardinalityQ>  
  <daml:hasClassQ rdf:resource=".../daml+oil#Thing" />  
</daml:Restriction>
```

# cardinality Restrictions

---

- ☞ Note that no unique name assumption:
- individual only instance of above class if it has 3 (provably) different friends
  - `maxCardinality` restrictions can lead to `sameIndividualAs` inferences

# RDF Syntax

---

- ➡ Syntax allows multiple properties/classes in single restriction

```
<daml:Restriction>  
  <daml:onProperty rdf:resource="#hasFriend" />  
  <daml:hasClass rdf:resource="#hasFriend" />  
  <daml:toClass rdf:resource="#Republican" />  
</daml:Restriction>
```

- ➡ Result may not be as expected

- at least one Republican friend and all friends Republicans
- at least one Republican friend **iff** all friends Republicans

- ➡ Bottom line: avoid such constructs! — use `intersectionOf 2` (or more) separate restrictions

# Class Expressions: Enumerations

---

## Existentially defined classes

- 👉 Class defined by listing members, e.g.:

```
<daml:Class>  
  <daml:oneOf rdf:parseType="daml:collection">  
    <rdf:Description rdf:about="#Italy">  
    <rdf:Description rdf:about="#France">  
  </daml:oneOf>  
</daml:Class>
```

# Class Expressions: Enumerations

---

- ☞ Strange properties compared to other classes
  - e.g., cardinality of class is known (2 in the above case)
- ☞ Powerful/useful but hard to deal with computationally
- ☞ Can sometimes substitute union of (primitive) classes, e.g.:

```
<daml:Class>  
  <daml:unionOf rdf:parseType="daml:collection">  
    <daml:Class rdf:about="#Italy"/>  
    <daml:Class rdf:about="#France"/>  
  </daml:unionOf>  
</daml:Class>
```

- but (max) cardinality inferences may be lost



# Class Expressions: Booleans

---

☞ Standard boolean constructors (intersection, union, complement) can be used to combine classes

☞ Boolean constructors are properties not a classes

- Class “wrapper” needed for nesting, e.g.:

```
<daml:Class rdf:ID="Woman" >
  <daml:intersectionOf rdf:parseType="daml:collection"
    <daml:Class rdf:about="#Person" />
    <rdfs:Class>
      <daml:complementOf rdf:resource="#Male" />
    </rdfs:Class>
  </daml:intersectionOf>
</daml:Class>
```

# Datatypes

---

Can use XMLS datatypes and values instead of classes and individuals:

- 👉 Domain of classes and datatypes considered disjoint
  - no object can be both class instance and datatype value
- 👉 Two types of property: `ObjectProperty` and `DatatypeProperty`
  - `ObjectProperty` used with classes/individuals
  - `DatatypeProperty` used with datatypes/values
- 👉 Can use arbitrary XMLS datatypes
  - built-in (primitive and derived), e.g., `xsd:decimal`
  - user defined/derived, e.g., sub-ranges
- 👉 Datatypes can be used in restrictions and as range of datatype properties
- 👉 Data values can be used in `hasValue` and in RDF “ground facts”

# Property Expressions

---

- ➡ Only property operator directly supported is `inverseOf`
- ➡ Other operators such as composition ( $\circ$ ) and union ( $\sqcup$ ) can sometimes be expanded out
  - $\exists(P1 \circ P2).C \equiv \exists P1.(\exists P2.C)$
  - $\forall(P1 \circ P2).C \equiv \forall P1.(\forall P2.C)$
  - $\exists(P1 \sqcup P2).C \equiv (\exists P1.C) \sqcup (\exists P2.C)$
  - $\forall(P1 \sqcup P2).C \equiv (\forall P1.C) \sqcap (\forall P2.C)$
- ➡ Can't capture/expand
  - intersection of properties
  - property expressions (except inverse) in cardinality restrictions, e.g.,  $\leq 1(P1 \circ P2)$  — but see “tricks of the trade”

# DAML+OIL Overview: Axioms

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
sameClassAs	$C_1 \doteq C_2$	Man $\doteq$ Human $\sqcap$ Male
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
samePropertyAs	$P_1 \doteq P_2$	cost $\doteq$ price
sameIndividualAs	$\{x_1\} \doteq \{x_2\}$	{President_Bush} $\doteq$ {G_W_Bush}
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq \neg$ Female
differentIndividualFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq \neg$ {peter}
inverseOf	$P_1 \doteq P_2^-$	hasChild $\doteq$ hasParent <sup>-</sup>
transitiveProperty	$P^+ \sqsubseteq P$	ancestor <sup>+</sup> $\sqsubseteq$ ancestor
uniqueProperty	$\top \sqsubseteq \leq 1P$	$\top \sqsubseteq \leq 1$ hasMother
unambiguousProperty	$\top \sqsubseteq \leq 1P^-$	$\top \sqsubseteq \leq 1$ isMotherOf <sup>-</sup>

# Class Axioms

---

Allow facts to be asserted w.r.t. classes/class expressions, e.g.,  
equivalence

👉 **All** class axioms can be transformed into `subClassOf`, e.g.:

$$C1 \equiv C2 \iff C1 \sqsubseteq C2 \text{ and } C2 \sqsubseteq C1$$
$$C1 \text{ disjointWith } C2 \iff C1 \sqsubseteq \neg C2$$

- but different forms may be useful for modelling and/or reasoning

👉 Most common axiom is `sub/sameClass` with name on l.h.s., e.g.:

`Triangle ≡ Polygon ⊔ =3 hasAngle.`

- sometimes called a **definition**
- can have as many definitions as we like
- no way to distinguish “main” definition

# Class Axioms

---

- 👉 multiple subClass axioms with same l.h.s. can be gathered together or separated, e.g.:

$$C1 \sqsubseteq C2, \quad C1 \sqsubseteq C3 \quad \iff \quad C1 \sqsubseteq C2 \sqcap C3$$

- but multiple equivalence axioms with same l.h.s. can **not** be gathered together

- 👉 In general, both sides can be arbitrary expressions, e.g.:

$$\text{Polygon} \sqcap =3 \text{hasSide} \sqsubseteq =3 \text{hasAngle}$$

- This feature is very powerful and allows many complex situations to be captured

# Class Axioms

---

- 👉 subClass axioms can be seen as a form of **rule**, e.g.:

$$C1(x) \leftarrow C2(x) \wedge P1(x, y) \wedge P2(y, z) \wedge C3(z)$$

is equivalent to

$$C2 \sqcap \exists P1.(\exists P2.C3) \sqsubseteq C1$$

- 👉 Synonyms can also be captured by asserting name equivalence, e.g.:

$$\text{Car} \equiv \text{Automobile}$$

# Class Axioms

---

- ➡ No requirement to “define” class before use
  - But good practice in general (for detecting typos etc.)
- ➡ Axioms can be directly (or indirectly) cyclical, e.g.:

$$\text{Person} \equiv \exists \text{hasParent. Person}$$

- Descriptive (standard FOL) semantics — not fixedpoint



# Property Axioms

---

Allow facts to be asserted w.r.t. properties/property expressions, e.g.:

$$\text{hasChild} \equiv \text{hasParent}^{-}$$

- 👉 Equivalence reducible to subProperty as for classes
- 👉 Multiple axioms/definitions etc. as for classes
- 👉 Can also assert that a property is **transitive**
  - Useful/essential for part-whole, causality etc.
  - Easier to handle computationally than transitive closure operator
  - Can combine with subPropertyOf to get similar effect, e.g.:

$$\text{directPartOf} \sqsubseteq \text{partOf} \textit{ and } \textit{transitive}(\text{partOf})$$

similar to

$$\text{directPartOf}^* \equiv \text{partOf}$$

- Can only be applied to object properties

# Property Axioms

---

- ➡ Symmetrical not directly supported but easily captured:

$$\text{hasNeighbour} \equiv \text{hasNeighbour}^{-}$$

- ➡ Reflexive cannot be captured

# Property Axioms

---

- ➡ Range/domain constraints equivalent to toClass restrictions on property/inverse subsuming Thing:

$$\begin{aligned}\text{range}(P, C) &\iff \text{Thing} \sqsubseteq \forall P.C \\ \text{domain}(P, C) &\iff \text{Thing} \sqsubseteq \forall P^-.C\end{aligned}$$

- ➡ Unique/unambiguous assertions equivalent to maxCardinality=1 restrictions on property/inverse subsuming Thing:

$$\begin{aligned}\text{uniqueProperty}(P) &\iff \text{Thing} \sqsubseteq \leq 1P \\ \text{unambiguousProperty}(P) &\iff \text{Thing} \sqsubseteq \leq 1P^-\end{aligned}$$

- ➡ Note that these are **very** strong statements

- restriction asserted w.r.t. Thing
- can result in “strange” (unexpected) inferences and/or compromise extensibility of ontology
- almost always better asserted locally (particularly range/domain)

# Individual Axioms

---

Allow facts to be asserted w.r.t. individuals, e.g., type

- 👉 RDF used for basic type/property assertions (Abox)

```
<Person rdf:ID="John" />  
<rdf:Description rdf:about="#John" > i.e.,  
  <hasParent:resource="#Mary" />  
</rdf:Description>
```

John  $\in$  Person,  $\langle$ John, Mary $\rangle \in$  hasParent

- 👉 Can state same facts using DAML+OIL oneOf, e.g.:

```
<daml:class>  
  <daml:oneOf rdf:parseType="daml:collection">  
    <rdf:Description rdf:about="#John">  
  </daml:oneOf>  
  <rdfs:subClassOf rdf:resource="#Person" />  
</daml:class>
```

# Individual Axioms

---

- 👉 Datatype properties relate individuals to data values
- 👉 Data values can be explicitly or implicitly typed, e.g.:

```
<rdf:Description rdf:about="#John">  
  <age>25</age>  
  <typedData><xsd:real rdf:value="3.14159" /></typedData>  
  <untypedData>1234</untypedData>  
</rdf:Description>
```

# Individual Axioms

---

👉 No unique name assumption

👉 But can assert equality or inequality of individuals, e.g.:

```
<rdf:Description rdf:about="#Clinton">
  <differentIndividualFrom:resource="#Hillary"/>
  <sameIndividualAs:resource="#BillClinton"/>
</rdf:Description>
```

👉 Can again use oneOf to capture such (in)equalities

```
<daml:class>
  <daml:oneOf rdf:parseType="daml:collection">
    <rdf:Description rdf:about="#Clinton">
  </daml:oneOf>
  <rdfs:sameClassAs rdf:resource="#BillClinton"/>
</daml:class>
```

# RDF Syntax

---

Slightly strange mixture of classes and properties, axioms and constructors

☞ Restrictions are classes

☞ Enumerations and booleans are properties

- implicit `sameClassAs` axiom, e.g.:

```
<daml:Class rdf:ID="NonPerson">
  <daml:complementOf rdf:resource="#Person" />
</daml:Class>
```

- have to be “wrapped” in an anonymous class to combine (e.g., with other booleans) or assert `subClassOf`

```
<daml:Class rdf:ID="Car">
  <rdfs:subClassOf>
    <daml:Class>
      <daml:complementOf rdf:resource="#Person" />
    </daml:Class>
  </rdfs:subClassOf>
</daml:Class>
```

# RDF Syntax

---

☞ Some constructors contain hidden axioms

- e.g., disjointUnionOf

```
<daml:Class rdf:about="#Person">  
  <daml:disjointUnionOf rdf:parseType="daml:collecti  
    <daml:Class rdf:about="#Man" />  
    <daml:Class rdf:about="#Woman" />  
  </daml:disjointUnionOf>  
</daml:Class>
```

includes **global** assertion about disjointness of Man and Woman

☞ Combined restrictions also hidden axioms



---

# Tricks of the Trade

# Using Property Hierarchy

---

- ➡ Common requirement is to construct class where 2 properties have same value
  - e.g., class of “happyPerson” whose spouse is the same individual as their best friend
  - Can achieve something similar using subPropertyOf and cardinality restrictions:

$$\begin{aligned} \text{hasSpouse} &\sqsubseteq \text{hasSpouseOrBestFriend} \\ \text{hasBestFriend} &\sqsubseteq \text{hasSpouseOrBestFriend} \\ \text{happyPerson} &\sqsubseteq =1 \text{ hasSpouse} \sqcap =1 \text{ hasBestFriend} \\ &\quad \sqcap \leq 1 \text{ hasSpouseOrBestFriend} \end{aligned}$$

- Note that all the properties must be locally unique
- ➡ Can also define bespoke part-whole hierarchy

# Inverse and oneOf

---

- 👉 oneOf is **very** powerful
- 👉 E.g., can be define so called “spy-point”
  - connected via some property to every object in domain

$$\text{Thing} \sqsubseteq \exists P.\{\text{spy-point}\}$$

- 👉 Combined with inverse can be used to fix (min/max) cardinality of domain, e.g.:

$$\{\text{spy-point}\} \sqsubseteq \leq 15 P^{-}$$

# General Axioms

---

General axioms (expressions on l.h.s.) are very powerful

☞ Can capture (some kinds of) rules, e.g.:

$$\text{period} = \text{lateGeorgian} \leftarrow \begin{array}{l} \text{culture} = \text{british} \\ \wedge \text{date} = 1760\text{--}1811 \end{array}$$

can be captured as an axiom:

$$\begin{array}{l} \exists \text{culture.british} \\ \sqcap \exists \text{date.1760--1811} \sqsubseteq \exists \text{period.lateGeorgian} \end{array}$$

☞ Can be computationally expensive

- should relativise as much as possible
- e.g., above axiom only relevant to furniture

# Other Useful Constructions

---

👉 Localised range/domain

$$C \sqsubseteq \forall P.D$$

$$C \sqcap \geq 1P \sqsubseteq D$$

👉 Localised unique/unambiguous

$$C \sqsubseteq \leq 1P$$

$$C \sqsubseteq \forall P.(\leq 1P1^-)$$

---

# Limitations

# What It Can't Do

---

DAML+OIL has many limitations, mostly designed to maintain decidability/computability/well-definedness

- 👉 Limited property constructors
  - e.g., no composition, transitive closure, product, ...
- 👉 Limited property types
  - transitive and symmetrical, but not reflexive
- 👉 Only collection type is set
  - e.g., no bags, lists
- 👉 Only unary and binary relations
- 👉 Restricted form of quantification (modal/guarded fragment)
- 👉 No comparison or aggregation of data values
- 👉 No defaults
- 👉 No variables (as in hybrid logics)
- 👉 ...

---

# Implementation challenges



# Implementation challenges

---

Even with existing language, challenges remain for would-be implementors

- ☞ Reasoning with oneOf is **hard**
  - decidable (contained in the C2 fragment of first order logic) but complexity increases from EXPTIME to NEXPTIME
  - no known “practical” algorithm
- ☞ Scalability
  - class consistency in EXPTIME even without oneOf
  - inverse properties cause particular difficulties
  - web ontologies may be **large**
- ☞ Other reasoning tasks
  - Querying
  - Explanation
  - LCS/matching