ulm university universität ulm

OWL
The Web Ontology Language

part I: overview

Pavel Klinov

In the next 90 mins

Intro

- why OWL?
- relationship to RDF(S) and logics (DLs)

Application areas and tools

Basics

- entities, expressions, axioms
- dealing with data values
- non-logical part: annotations, imports, and versions

So why Semantic Web needs OWL?

First, we've got RDF

- A simple graph language to express facts (LD)
- A simple data model + low-level data integration tools

So why Semantic Web needs OWL?

First, we've got RDF

- A simple graph language to express facts (LD)
- A simple data model + low-level data integration tools

No schema? But we have RDFS!

- A lightweight schema, good for simple vocabularies
- Some simple inferencing (transitivity of rdfs:subClassOf)

RDF(S) not quite sufficient

Schemas are often just too weak

- Can say: :hasWife rdfs:domain :Woman (rdfs:range :Woman)
- Cannot say: :Peter :hasWife some :Woman (only :Woman)

RDF(S) not quite sufficient

Schemas are often just too weak

- Can say: :hasWife rdfs:domain :Woman (rdfs:range :Woman)
- Cannot say: :Peter :hasWife some :Woman (only :Woman)

Reasoning is weak but very hard

- NP-complete without negation, disjunction, etc.

RDF(S) not quite sufficient

Schemas are often just too weak

- Can say: :hasWife rdfs:domain :Woman (rdfs:range :Woman)
- Cannot say: :Peter :hasWife some :Woman (only :Woman)

Reasoning is weak but very hard

- NP-complete without negation, disjunction, etc.

So we need a language(s) that:

- Provides adequate balance between expressivity and computational complexity
- “pay-as-you-go” behavior

RDF(S) not quite sufficient

Schemas are often just too weak

- Can say: :hasWife rdfs:domain :Woman (rdfs:range :Woman)
- Cannot say: :Peter :hasWife some :Woman (only :Woman)

Reasoning is weak but very hard

- NP-complete without negation, disjunction, etc.

So we need a language(s) that:

- Provides adequate balance between expressivity and computational complexity
- “pay-as-you-go” behavior

That language is called OWL 2

Application areas

Vocabulary-centric applications

- manage complex terminologies
  (in a machine-processable way)
- share terminology across applications

Data-centric applications

- lightweight reasoning over tons of data
- data integration

Terminology management

Example: medical informatics

- Terminologies are huge
  - ICD: ~100K medical codes
  - SNOMED CT: >300K classes
- Applications are different
  - medical diagnosis tools
  - electronic medical records
  - learning and statistical analysis tools

All must agree on the meaning of the terms

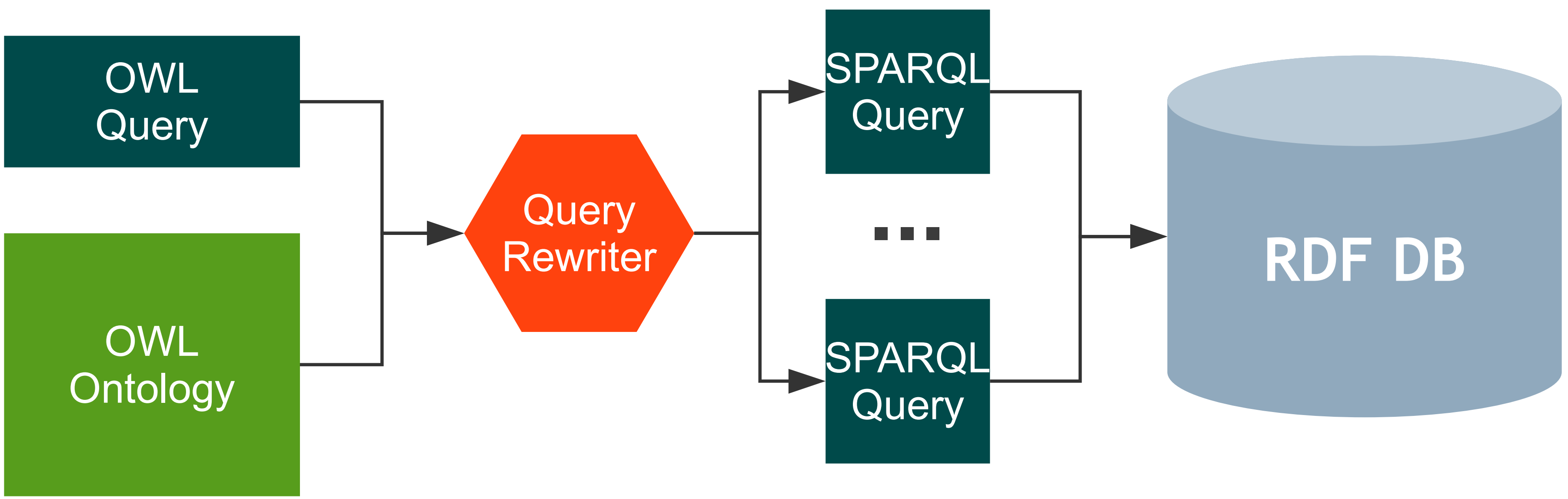
Scalable schema reasoning is key

- e.g. for quality assurance

Data-centric apps

Mostly about querying loads of data

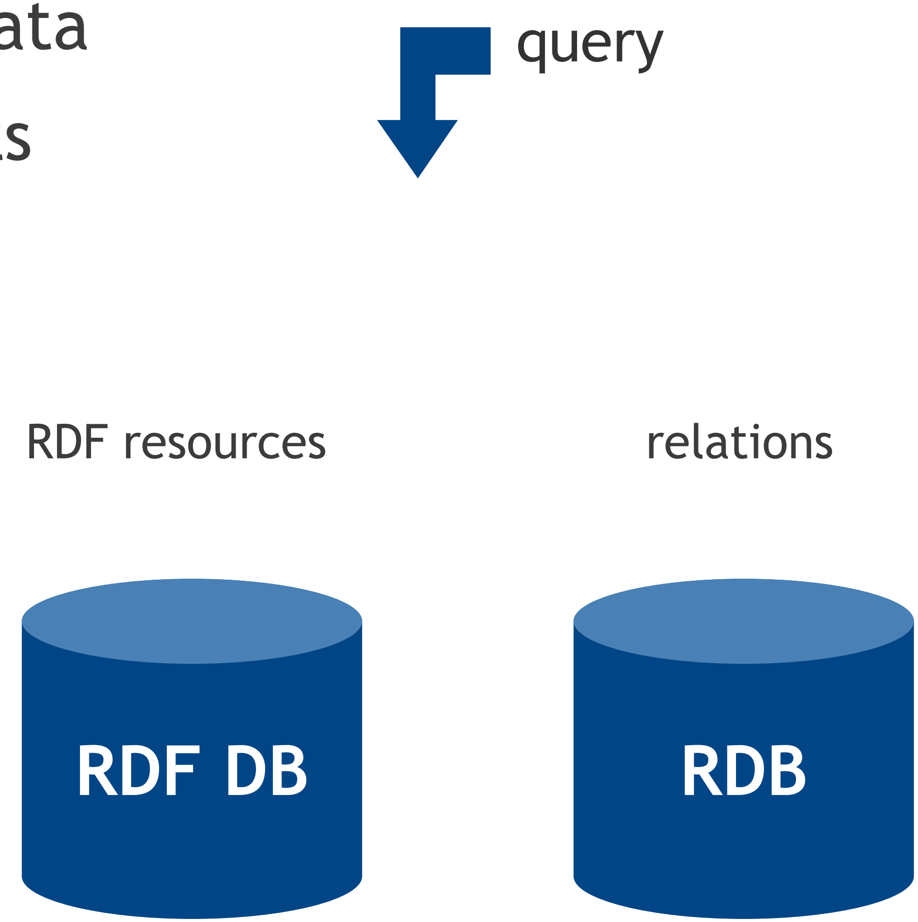
- w.r.t. some (simple) schema
- on top of RDF (or SQL) database



Data integration

Data sources are often heterogeneous

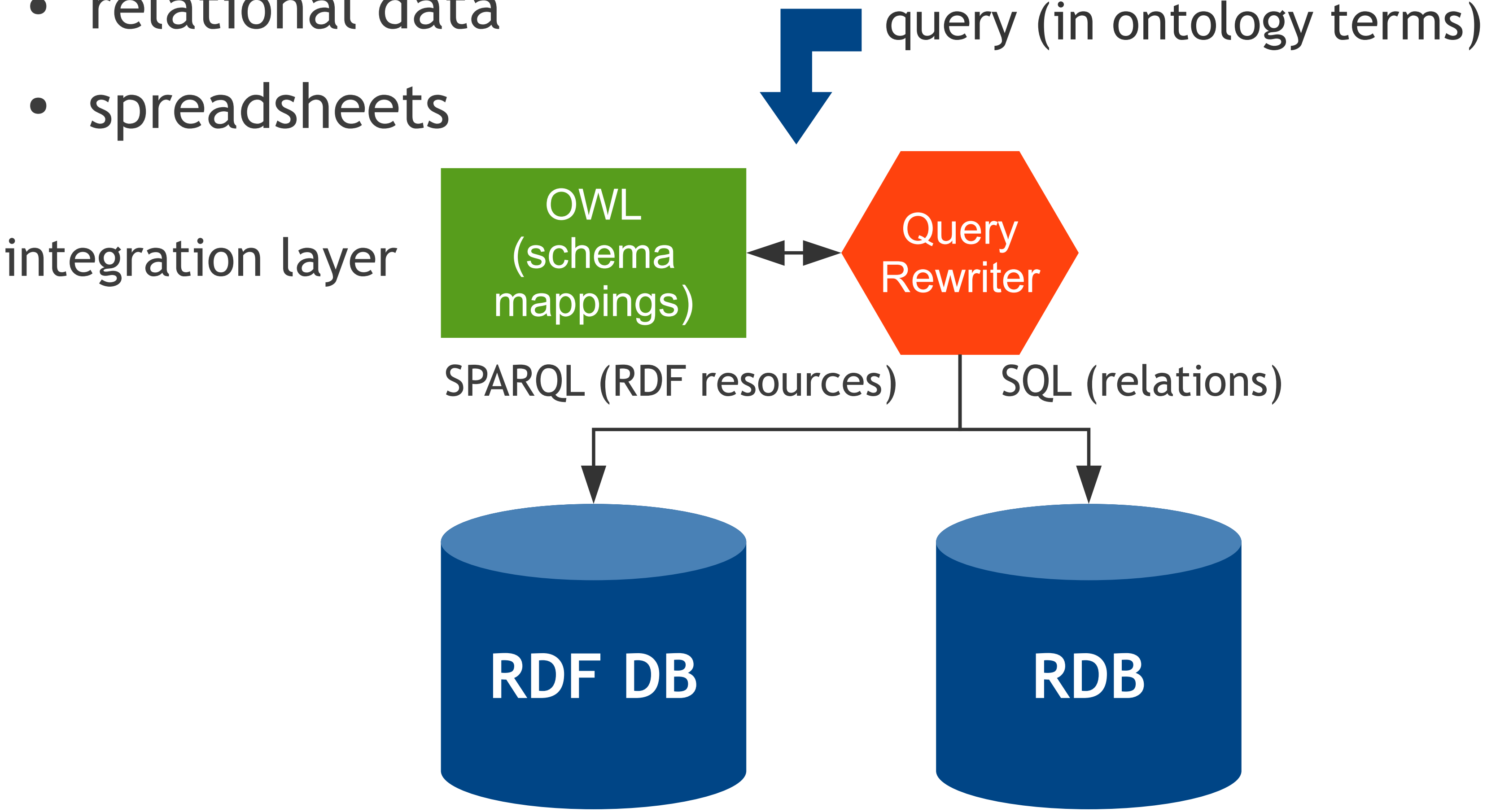
- RDF data
- relational data
- spreadsheets



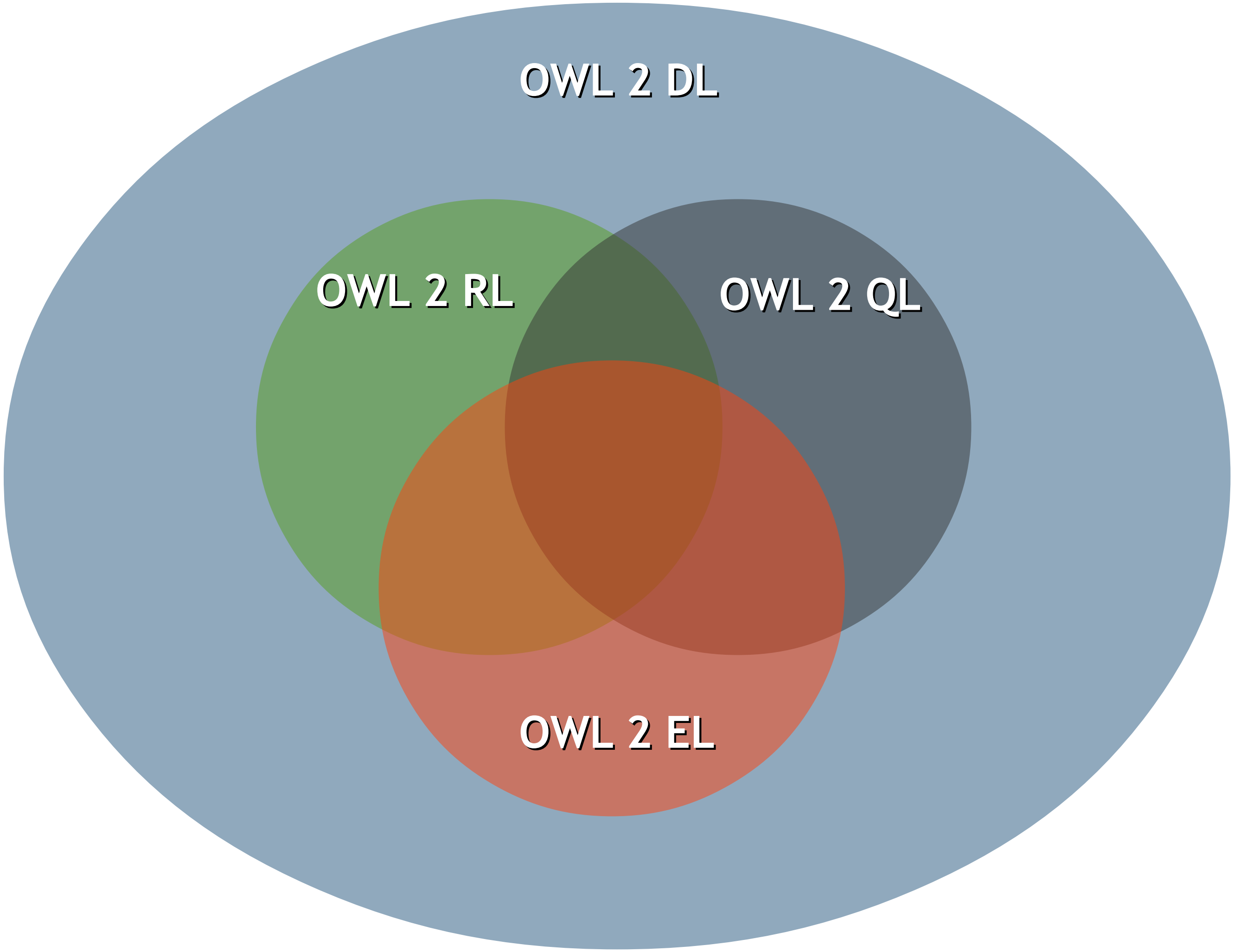
Data integration

Data sources are often heterogeneous

- RDF data
- relational data
- spreadsheets



One size does not fit all!



Tools

Reasoners
- DL: Pellet, FaCT++, HermiT, RACER
- Lightweight: CEL/jCEL/ELK/Quest

Semantic databases
- Virtuoso, Stardog, OWLIM, Oracle 11
- not always fully implement profiles

- APIs: OWL API, RDF-based APIs (Jena, Sesame, etc.)
- Data integration (PDQ)
- Matchers, editors, debuggers, visualizers...

Extended RDF or logic?

OWL as RDF extension

- Every OWL ontology can be expressed as RDF graph (the other way is trickier)
- a semantically compatible RDF graph

Extended RDF or logic?

OWL as RDF extension

- Every OWL ontology can be expressed as RDF graph (the other way is trickier)
- a semantically compatible RDF graph

OWL as a logic with a Web-friendly syntax

- OWL ontology is a DL knowledge base
- with a DL semantics

Extended RDF or logic?

OWL as RDF extension

- Every OWL ontology can be expressed as RDF graph (the other way is trickier)
- a semantically compatible RDF graph

OWL as a logic with a Web-friendly syntax

- OWL ontology is a DL knowledge base
- with a DL semantics

The views are compatible to a certain extent

We adopt the second view in this lecture

Schema vs Data

Think RDB

- schema defines structure (tables, keys, attributes)
- data specifies facts

Schema vs Data

Think RDB

- schema defines structure (tables, keys, attributes)
- data specifies facts

OWL

- schema (TBox) statements describe the domain
- data (ABox) statements express facts (like RDF)
- both are called axioms
- TBox + ABox is called ontology

Modeling example

Family

- parent, children
- cousins, aunts, uncles, nephews, etc.
- pets

Need to model to define terms unambiguously

- to manage data
- to make apps understand the data
- to make sure different apps agree on terms

A simple example (TBox, ABox)

TBox: conceptual modeling

- a parent is a mother or a father
- father and mother are disjoint concepts
- every person must have one parent of each kins
- your parents' parents are your grandparents

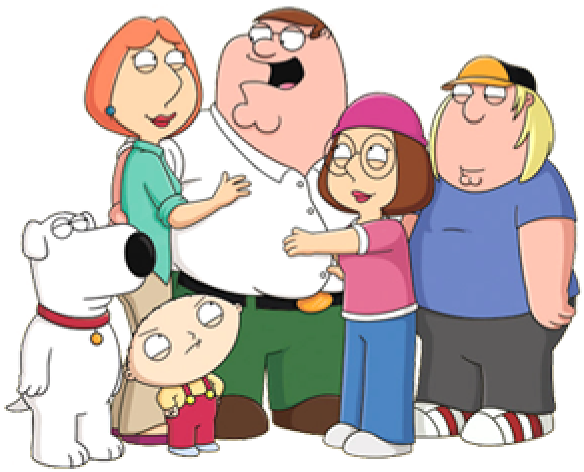
A simple example (TBox, ABox)

TBox: conceptual modeling

- a parent is a mother or a father
- father and mother are disjoint concepts
- every person must have one parent of each kins
- your parents' parents are your grandparents

ABox: a specific family

- Peter is a father, Lois is a mother
- Peter and Lois are parents of Chris, Meg, and Stewie
- Pewterschmidts are parents of Lois



Where the analogies stop...

OWL ontologies are not databases

- DB are closed-world collections of facts: either explicitly true or false (NAF)

Where the analogies stop...

OWL ontologies are not databases

- DB are closed-world collections of facts: either explicitly true or false (NAF)
- ontologies are open-world: things can be explicitly true, implicitly true, false, or unknown
  - explicitly true: Peter is a father
  - implicitly true: Chris, Meg, and Stewie have grandparents
  - false: Lois is a father
  - unknown: Chris is a parent

Where the analogies stop...

OWL ontologies are not databases

- DB are closed-world collections of facts: either explicitly true or false (NAF)
- ontologies are open-world: things can be explicitly true, implicitly true, false, or unknown
  - explicitly true: Peter is a father
  - implicitly true: Chris, Meg, and Stewie have grandparents
  - false: Lois is a father
  - unknown: Chris is a parent
- no unique name assumption: Chris, Meg, and Stewie could all denote the same person

Where the analogies stop...

OWL is not a programming language

- modeling is declarative, describes what's true
- no procedural semantics (triggers, slots, etc.)
- doesn't specify how to infer what's true

Where the analogies stop...

OWL is not a programming language

- modeling is declarative, describes what's true
- no procedural semantics (triggers, slots, etc.)
- doesn't specify how to infer what's true

OWL is not a schema language

- can't impose syntactic constraints on documents (e.g. like in XML Schema)
- example: can't require that parent axioms are syntactically present

On OWL syntaxes

There are many:

- RDF-native: RDF/XML, Turtle, N3, etc.
  all describe triples
- OWL-native: OWL/XML, Functional, Manchester
  all describe axioms

This lecture uses the Functional Syntax

- avoid OWL axiom to RDF triples mapping
- avoid XML verbosity

Axioms, Entities, and Expressions

Entities

Main building blocks: classes, properties, individuals (all denoted with IRIs)

- Individuals: specific objects

  :Peter, :Lois, etc.
- Classes (concepts): sets of individuals

  :Family, :Parent
- Properties (roles): sets of pairs of individuals

  :marriedTo, :childOf

Entities

Main building blocks: classes, properties, individuals (all denoted with IRIs)

- Individuals: specific objects

  :Peter, :Lois, etc.
- Classes (concepts): sets of individuals

  :Family, :Parent
- Properties (roles): sets of pairs of individuals

  :marriedTo, :childOf

Entities need to be declared in OWL 2 DL

Declaration (ObjectProperty (:hasParent))

Class expressions (CE)

Classes with a IRI are called named or atomic

:Person, :Parent, ...

owl:Thing (⊤) and owl:Nothing (⊥) are predefined

Can be combined into class expressions

- expressions don't have IRIs
- still interpreted as sets
- propositional and non-propositional

Property expressions

Named properties

- identified with IRI
- owl:topObjectProperty and owl:bottomObjectProperty
- object properties and data properties

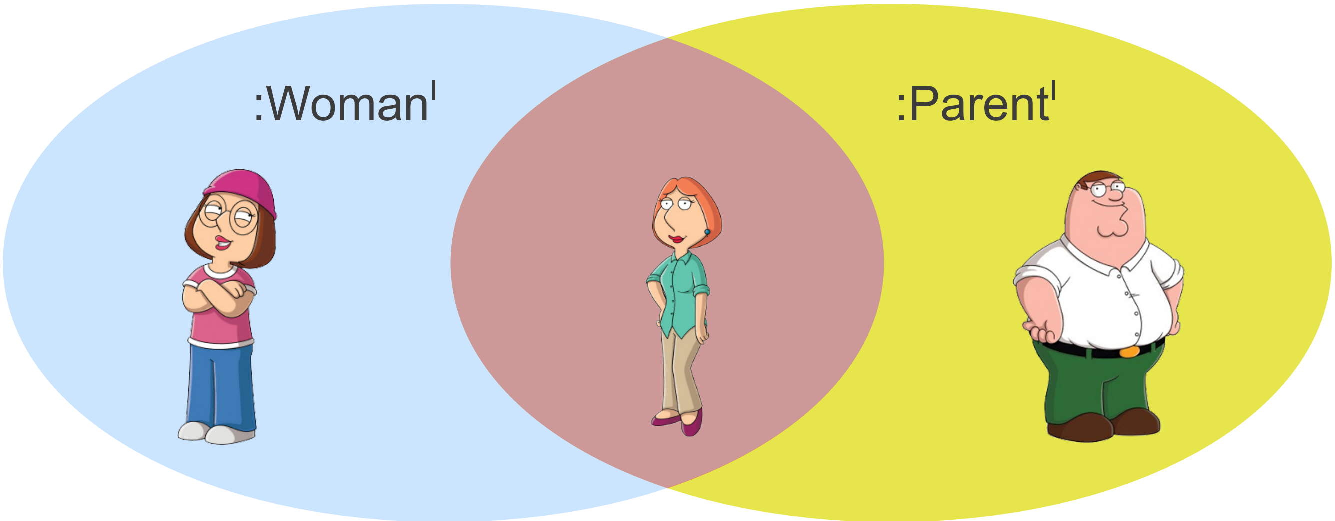
Property expressions

- no IRIs
- also interpreted as relations

CE: boolean constructors

OWL 2 DL is a propositionally complete language

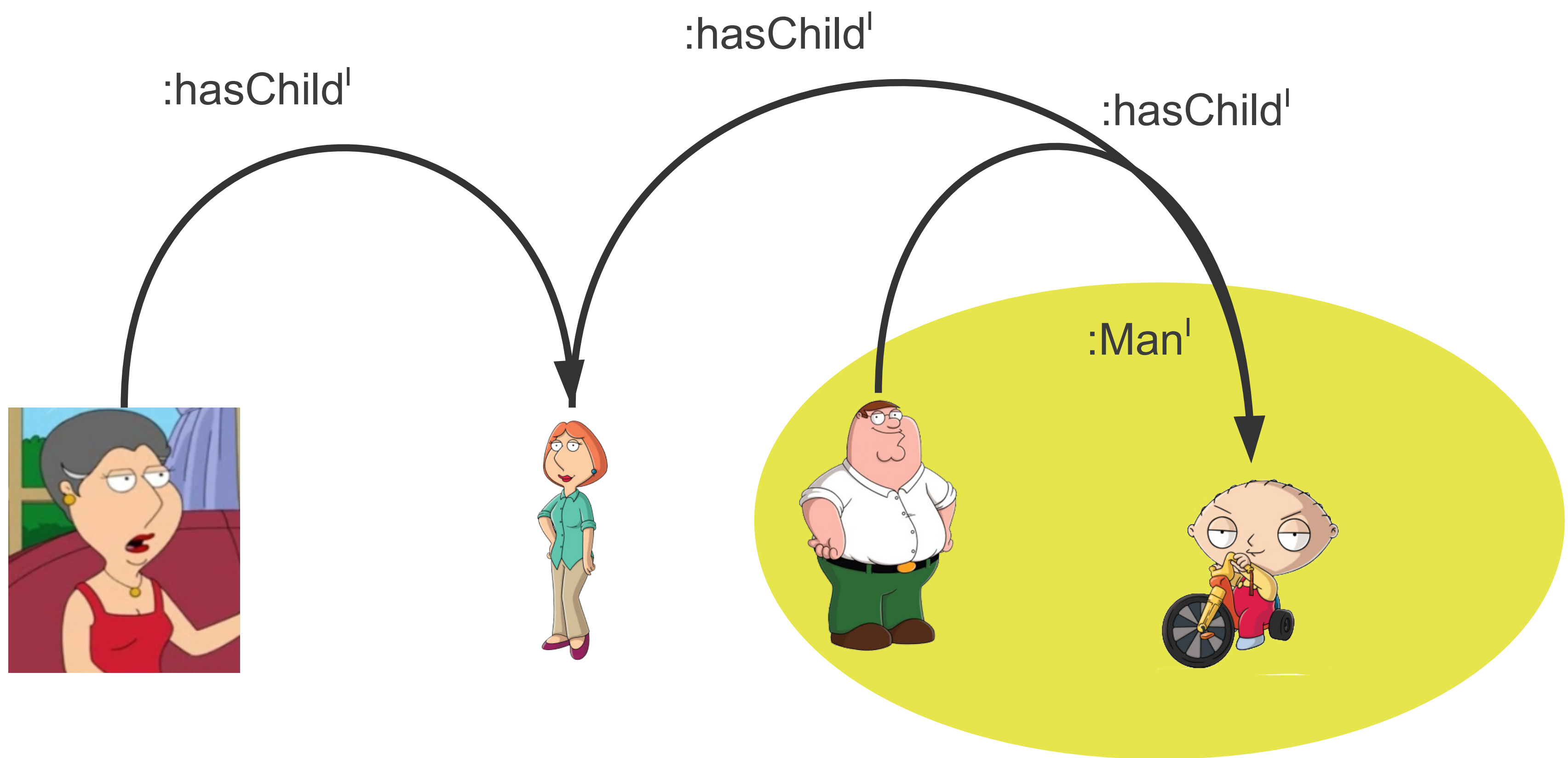
- intersection: ObjectIntersectionOf(:Woman :Parent)
- union: ObjectUnionOf(:Mother :Father)
- complement: ObjectComplementOf(:Parent : Mother)



CE: restrictions on properties

Existentials:

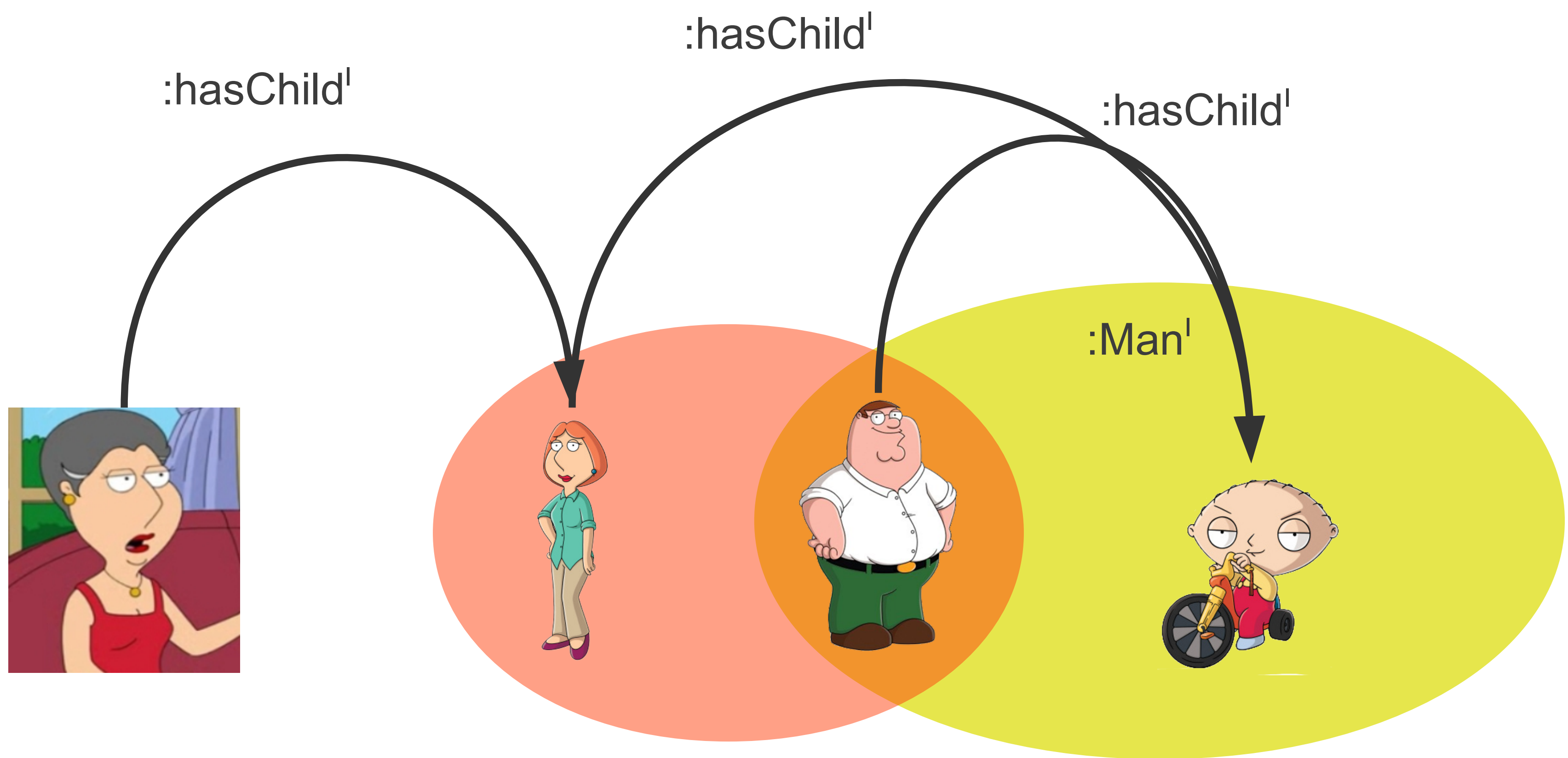
ObjectSomeValuesFrom(:hasChild :Man)



CE: restrictions on properties

Existentials:

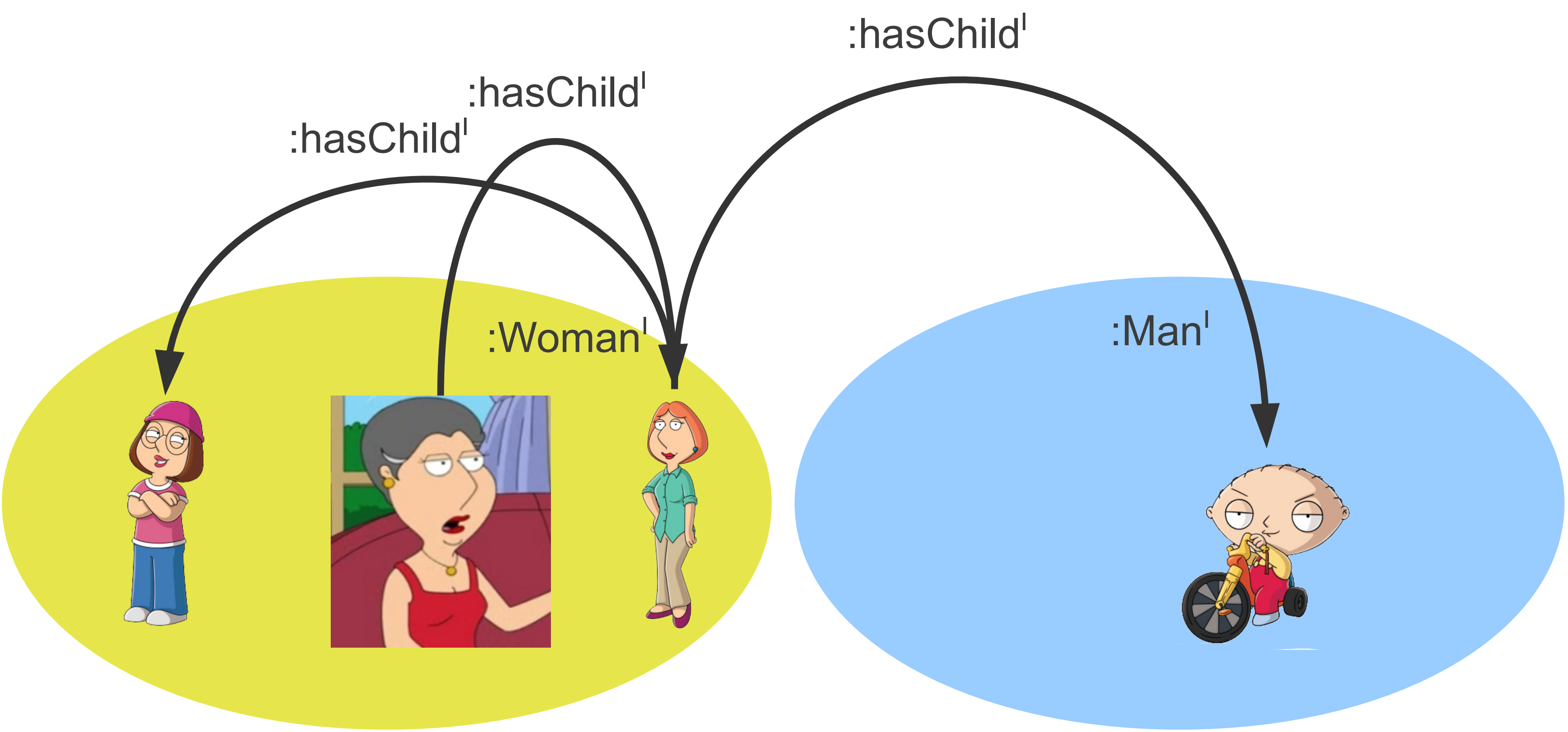
ObjectSomeValuesFrom(:hasChild :Man)



CE: restrictions on properties

Universals:

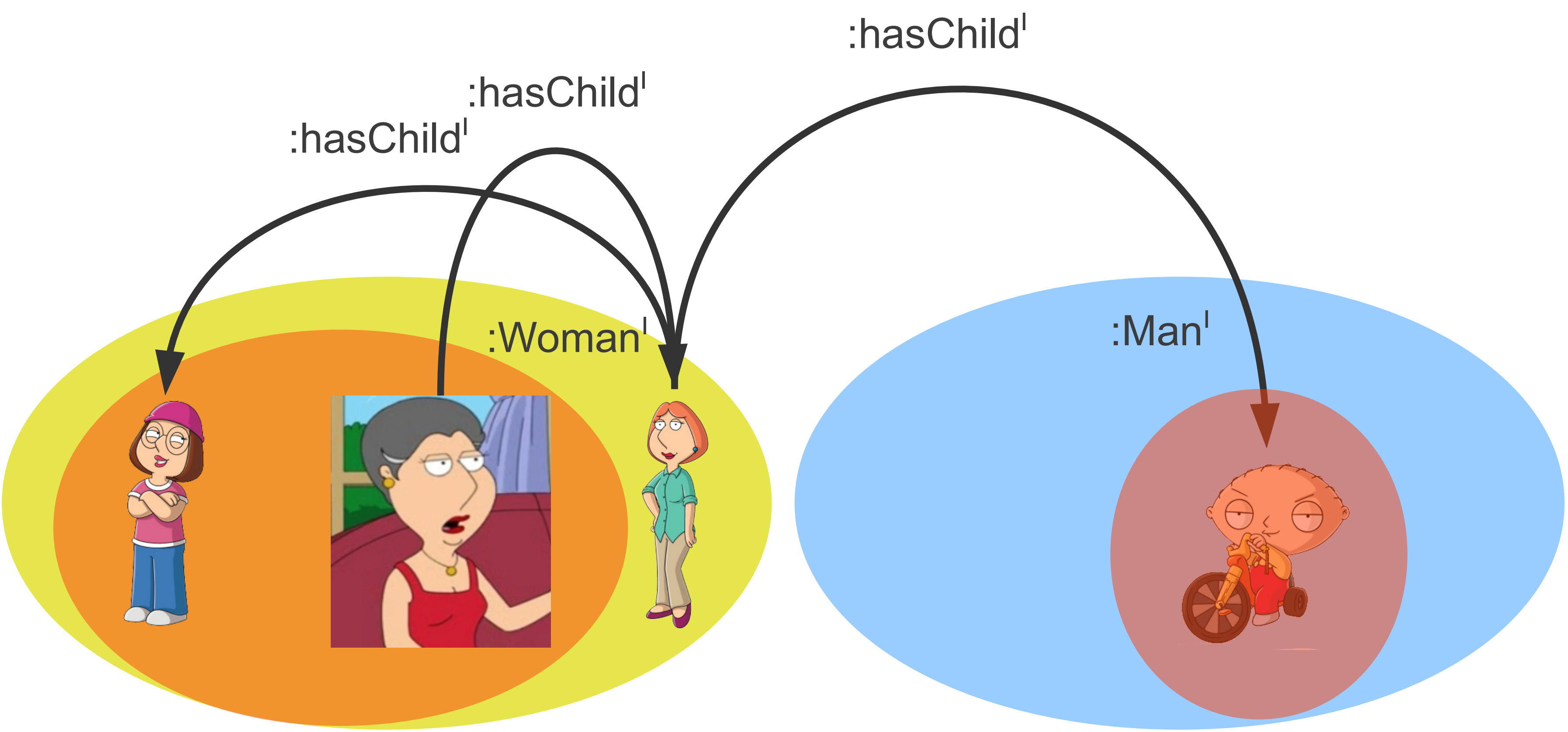
ObjectAllValuesFrom(:hasChild :Woman)



CE: restrictions on properties

Universals:

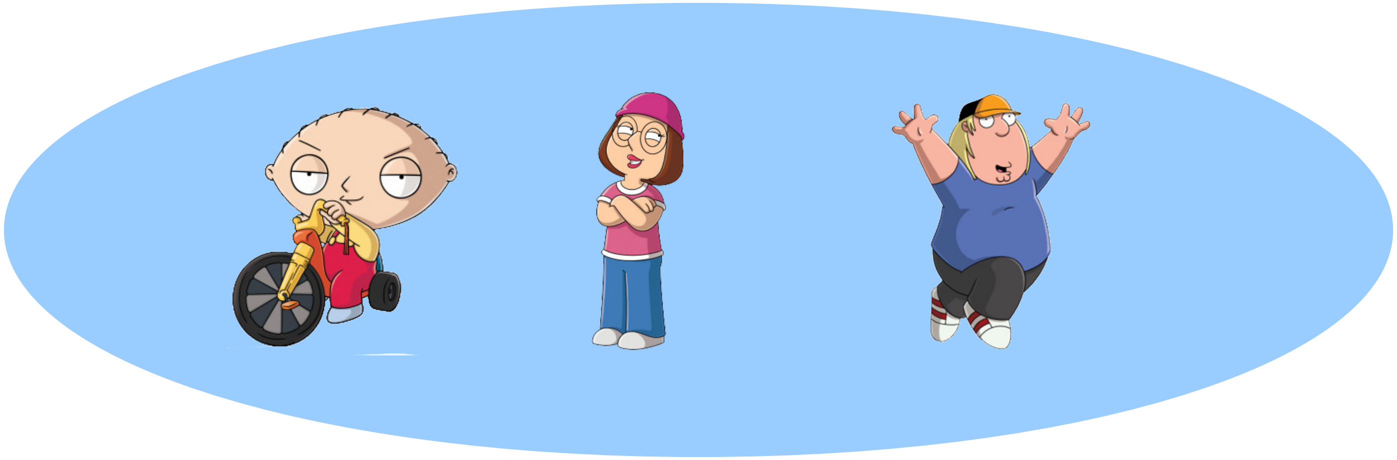
ObjectAllValuesFrom(:hasChild :Woman)



Nominals classes

Sometimes you just want to enumerate things

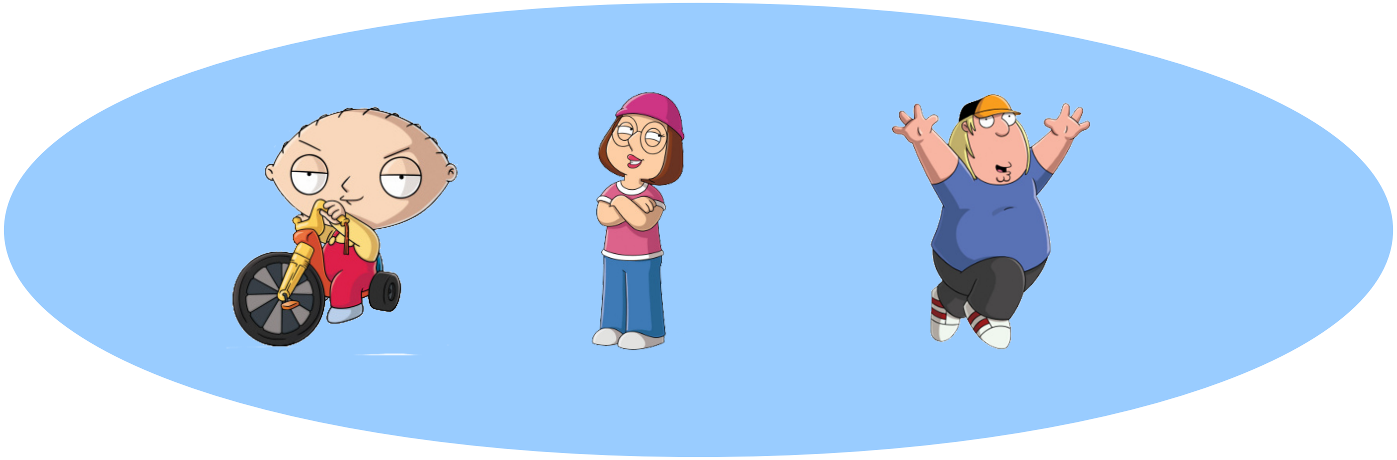
ObjectOneOf(:Chris :Meg :Stewie)



Nominals classes

Sometimes you just want to enumerate things

ObjectOneOf(:Chris :Meg :Stewie)



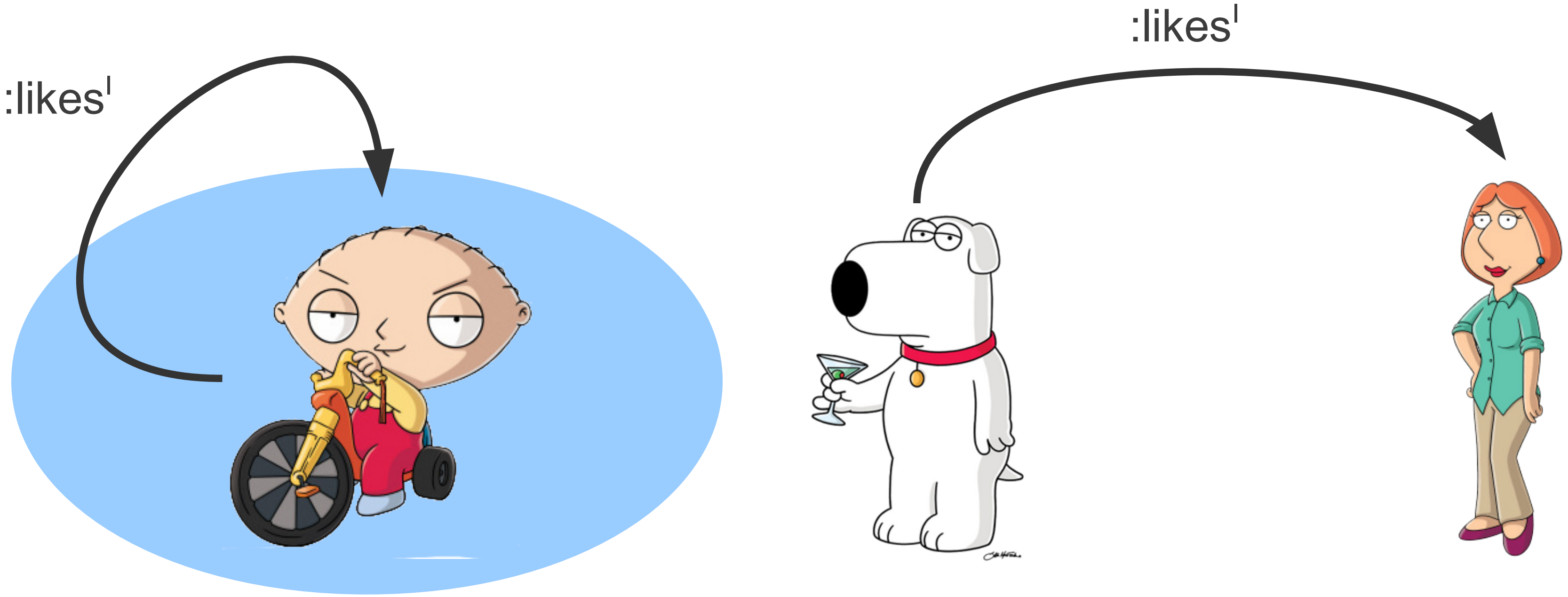
Does it mean that the class

- contains exactly 3 objects?
- at least 3? at most 3?

Self-restrictions

Can define a class of objects related to itself

ObjectHasSelf(:likes)



Cardinality restrictions

ObjectMinCardinalityFrom (2 :hasChild owl:Thing)

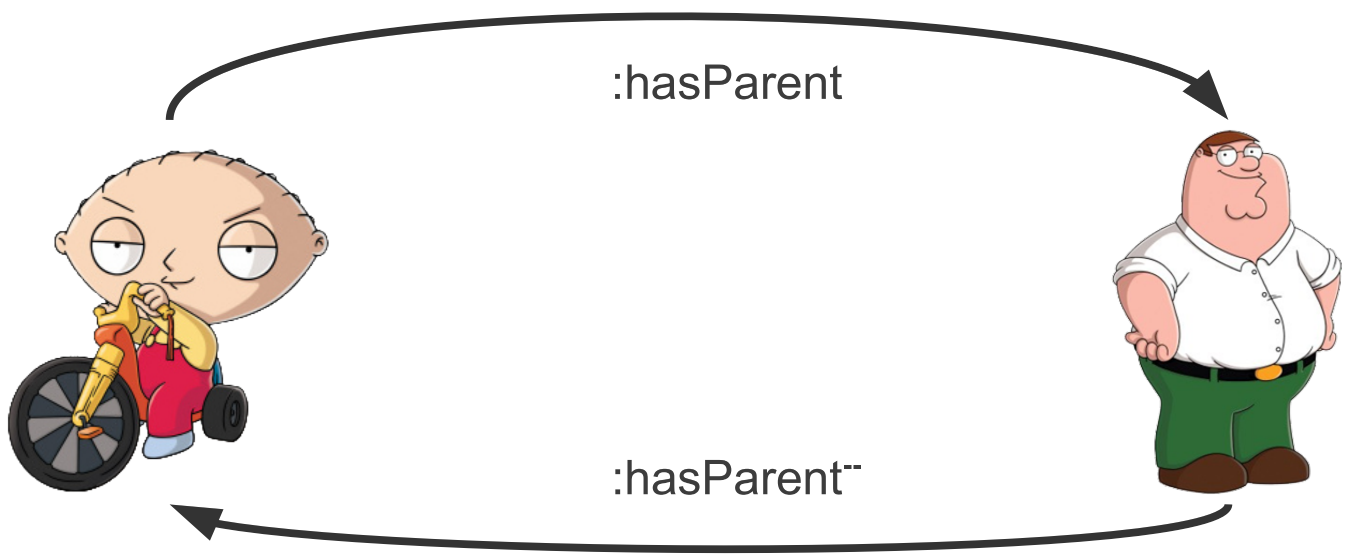
ObjectMaxCardinalityFrom (2 :hasParent :Parent)

Property restrictions

Inverse properties

ObjectInverseOf(:hasChild)

interpreted as inverse relations

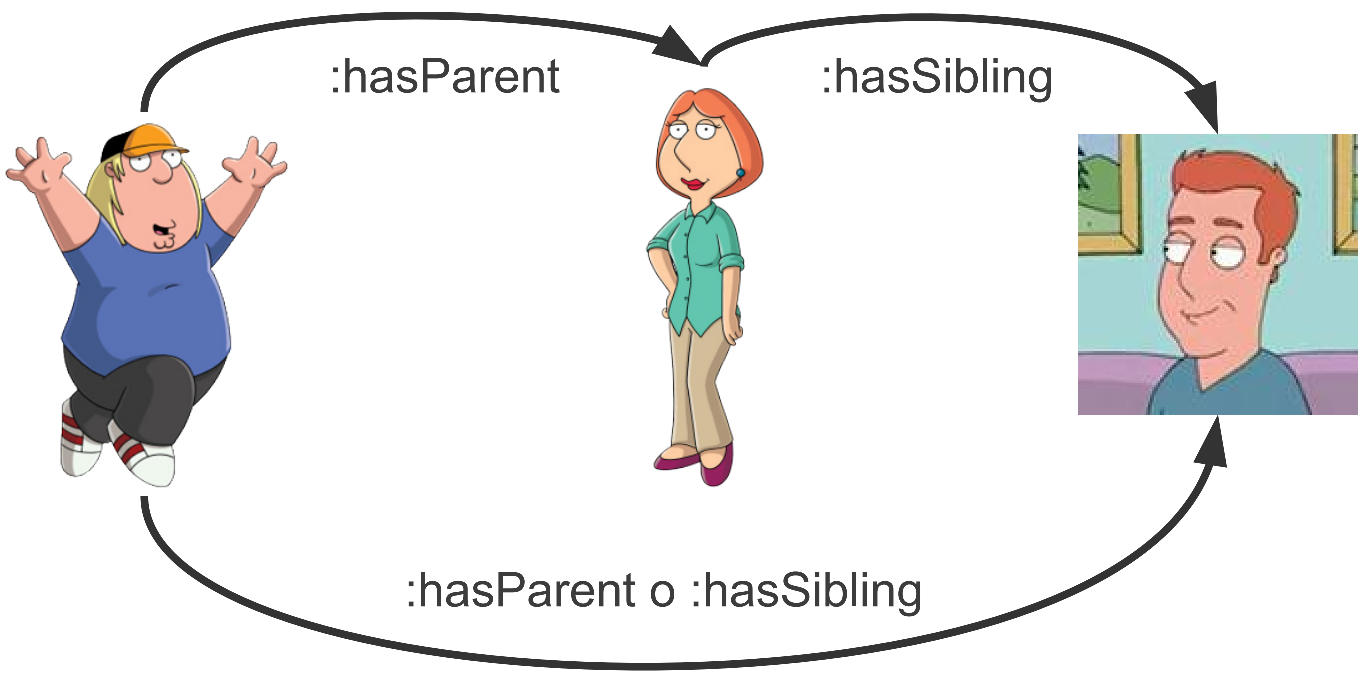


Property restrictions

Property chains

ObjectPropertyChain(:hasParent :hasSibling)

interpreted as compositions of relations



Axioms

TBox axioms

- relationships between classes (e.g. inclusion)
- relationships between properties

ABox axioms

- class membership
- property membership
- individual equality/inequality

Class inclusions

```
SubClassOf( :Woman :Person )
SubClassOf(
    :Grandfather
    ObjectIntersectionOf( :Man :Parent ))
```

Class equivalence

EquivalentClasses(

:Mother

ObjectIntersectionOf(:Woman :Parent))

- all mothers are women and parents
- vice versa

Class disjointness

DisjointClasses(:Father :Mother)

no instance of A is an instance of B (and vice versa)

Property axioms

Property inclusions

simple: SubObjectPropertyOf(:hasWife :hasSpouse)

chains:

SubObjectPropertyOf(

ObjectPropertyChain(

:hasParent :hasParent) :hasGrandparent)

Property axioms

FunctionalObjectProperty(:hasMother)

InverseFunctionalObjectProperty(:motherOf)

ReflexiveObjectProperty(:likes)

IrreflexiveObjectProperty(:hates)

TransitiveObjectProperty(:partOf)

SymmetrycObjectProperty(:hasSpouse)

AsymmetricObjectProperty(:hasChild)

The feature set isn't minimal

Existentials and universals

- ObjectSomeValuesFrom(:hasChild :Person)
- ObjectAllValuesFrom(:hasChild :Person)

Class equivalence and disjointness (trivial)

Transitivity?

The feature set isn't minimal

Existentials and universals

- ObjectSomeValuesFrom(:hasChild :Person)
- ObjectAllValuesFrom(:hasChild :Person)

Class equivalence and disjointness (trivial)

Transitivity?

Or even ABoxes?

SubClassOf(ObjectOneOf(:Stewie) :Person)

ClassAssertion(:Person :Stewie)

Where are we?

Parts we've covered

- entities, class expressions
- object properties

Next

- data types and data properties
- very similar to classes and object properties!

Later: non-logical part

- imports
- versions
- annotations

OWL and data values

OWL is a two-sorted language

- The abstract domain
  - classes, properties, named objects

    ObjectPropertyAssertion(:fatherOf :Peter :Meg)

OWL and data values

OWL is a two-sorted language

- The abstract domain
  - classes, properties, named objects

    ObjectPropertyAssertion(:fatherOf :Peter :Meg)
- The concrete (or data) domain
  - strings, numbers, dates, etc.

    DataPropertyAssertion(:hasAge :Meg “17”^^xsd:integer)

Abstract and data domains

Abstract domain: Δ

- non-empty and arbitrary
- finite or infinite

Data domain Δ

- a superset of standard value sets (e.g., integers)
- fixed!

The domains are disjoint

The abstract world of logic

For developing theories about the world

- modelers often cautious and pedantical
- Open World Assumption, no Unique Name Assumption
- instances are defined only by axioms

The abstract world of logic

For developing theories about the world

- modelers often cautious and pedantical
- Open World Assumption, no Unique Name Assumption
- instances are defined only by axioms

Makes sense because

- usually better to under-model than to over-model
  - under-modeling loses entailments
  - over-modeling introduces errors
- gives extra flexibility

The concrete, data world

For re-using standard data theories

- have excellent theories about numbers, etc.
- know how to use them, how to compute with them
- don't need custom, ill-made integer ontologies!
- don't need UNA

The concrete, data world

For re-using standard data theories

- have excellent theories about numbers, etc.
- know how to use them, how to compute with them
- don't need custom, ill-made integer ontologies!
- don't need UNA

Datatypes fix what we know about, e.g., integers

- “4”^^xsd:integer and “6”^^integer aren't equal
- because all names have fixed meaning
  - somewhat like owl:'Thing
  - except that the concrete domain is always the same

Connecting the worlds

Data properties

- map abstract individuals to concrete data values
- DataSomeValueFrom(:hasWeight

  :Peter “100”^^xsd:integer)

Semantics

- interpreted as subsets of $\Delta x \Delta_d$

Data axioms

Axioms (mostly as for object properties)

- equivalence, inclusion, disjointness
- domains and ranges
- assertions
- functionality

But

- no chains (even transitivity)
- no inverses, reflexivity, symmetry
- can't go the other way or break the separation

More on fixed semantics

Example:

DataPropertyAssertion(:hasAge
:Meg “17”^^xsd:integer)

DataPropertyAssertion(:hasAge
:Meg “16”^^xsd:integer)

FunctionalDataProperty(:hasAge)

This is inconsistent

Try to formalize this logically!

- Remember UNA
- DifferentIndividuals(17 16)?

More on datatypes

Datatype: a kind of data values (integers, strings)

- IRI
- lexical space: “str”, “1”^^xsd:integer, “01”^^xsd:integer
- value space: “str”, 1
- facet spaces: pairs (F, v), mapped to a subset of VS
  - F: constraining facet
  - v: constraining value
  - (xsd:minExclusive , “30”^^xsd:integer)

Datatype map: a particular set of datatypes

- for a language
- for a particular tool (reasoner)

The OWL 2 datatype map

XSD datatypes

- decimals, integers (and subtypes)
- xsd:float
- xsd:double
- strings (subtypes of rdf:PlainLiteral)

Nuances:

- decimals and integers are subtypes of owl:real
- which is pairwise disjoint with xsd:float and xsd:double

```
DataPropertyRange( a:hasAge xsd:integer )
DataPropertyAssertion( a:hasAge a:Meg "17"^^xsd:double )
```

Data ranges

Abstract world analogue: class expressions

Can define custom datatypes based on standard ones

DataRange

- Datatype (like xsd:integer)
- DataUnionOf, DataIntersectionOf, DataComplementOf

  DataUnionOf(xsd:string xsd:integer)
- DataOneOf

  DataOneOf("1"^^xsd:integer “2.5”^^xsd:double)
- DatatypeRestriction

Datatype restrictions

Can constrain a datatype using facets

DatatypeRestriction(DT F_1 v_1 ... F_2 v_n)

Example:

DatatypeRestriction(xsd:integer

xsd:minInclusive "5"^^xsd:integer xsd:maxExclusive "10"^^xsd:integer)

contains only 5, 6, 7, 8, 9

facets are combined conjunctively

Datatype definitions

Can assign names to custom (restricted) datatypes

DatatypeDefinition(DT DR)

Example:

DatatypeDefinition(

:email

DatatypeRestriction(xsd:string xsd:pattern "..."))

Now can use :email in data axioms:

DataPropertyRange(:hasEmail :email)

Identifying abstract individuals

What if we need to identify objects by their “attributes”?

For object property values

- use inverse functional properties

  InverseFunctionalProperty(:hasName)

Identifying abstract individuals

What if we need to identify objects by their “attributes”?

For object property values

- use inverse functional properties

  InverseFunctionalProperty(:hasName)

Problems:

- global inverse functionality often undesirable

  (name's only unique within the Griffin family)
- how about data properties?

  no inverse functional data properties

Keys

HasKey(CE (OPE_1 … OPE_m) (DPE_1 … DPE_n))

This says that:

- if two named individuals of CE coincide on …
- … values of all object properties …
- … and values of all data properties, then
- the individuals are identical

Example:

- HasKey(:GriffinFamily (:hasName) ())
- HasKey(owl:Thing () (:hasTaxId))

Where are we?

Covered the logical part

- abstract part (class expressions, object properties)
- data part (datatypes, data ranges, data properties)
- axioms

Next: non-logical part

- imports
- versions
- annotations

Imports

Ontologies are meant to be reusable

OWL supports knowledge reuse via importing

```
Ontology(<http://fox.com/familyguy>
Import( <http://example.org/families.owl> ))
```

Imports

Ontologies are meant to be reusable

OWL supports knowledge reuse via importing

```
Ontology(<http://fox.com/familyguy>
Import( <http://example.org/families.owl> ))
```

Particularly important in HCLS, biology, etc.

- pros: reuse other people efforts
- cons: can be too heavyweight

solution: modularity (on Friday)

Versions

Ontologies are identified with a IRI

but also may have a version IRI to distinguish versions

Ontology(<http://fox.com/familyguy>

<http://fox.com/familyguy/2.0>

Why?

- ontologies are like public APIs (for your or shared data)
- changing your ontology may break others

Annotations

Not all content has to be logical

Meta-information

- author info
- axiom labels, comments
- provision

Modeling these on the logical level is unnecessary

- aren't statements about the domain
- statements about statements about the domain!

OWL 2 provides annotation support for these

Annotations

Subjects: ontologies or entities

Assertion: <annotationProperty, annotationValue>

Values: IRIs, literals, or individuals

Examples:

- AnnotationAssertion(rdfs:label a:Peter
  "Represents the main character from Family Guy")
- Ontology(<http://fox.com/familyguy>
  Annotation(rdfs:label "A Family Guy ontology")

Often useful for i18n

End of the basics

questions?