# OWL 2 – Theory and Practice

**Bernardo Cuenca Grau**
University of Oxford
UK

**Pascal Hitzler**
Kno.e.sis Center
Wright State University
Dayton, OH, USA

**Birte Glimm**
University of Oxford
UK

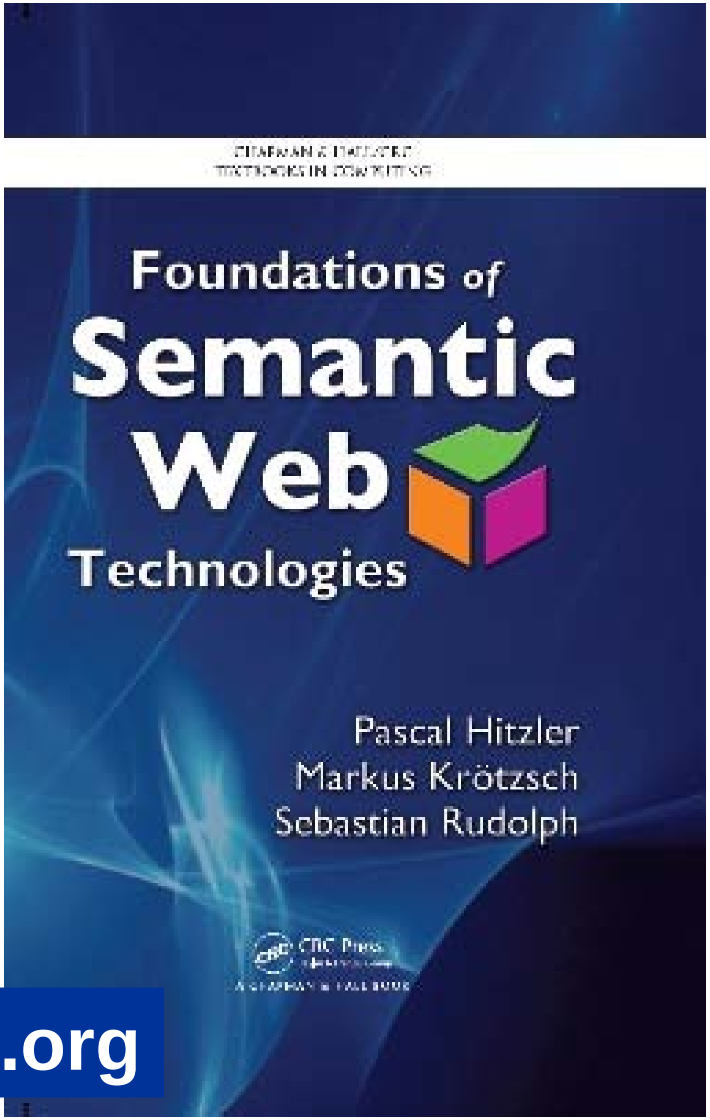
**Hector Perez-Urbina**
Clark & Parsia, LLC

# Textbook

**Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph**

**Foundations of Semantic Web Technologies**
**Chapman & Hall/CRC, 2009**

**Grab a flyer!**




**http://www.semantic-web-book.org**

# Slides

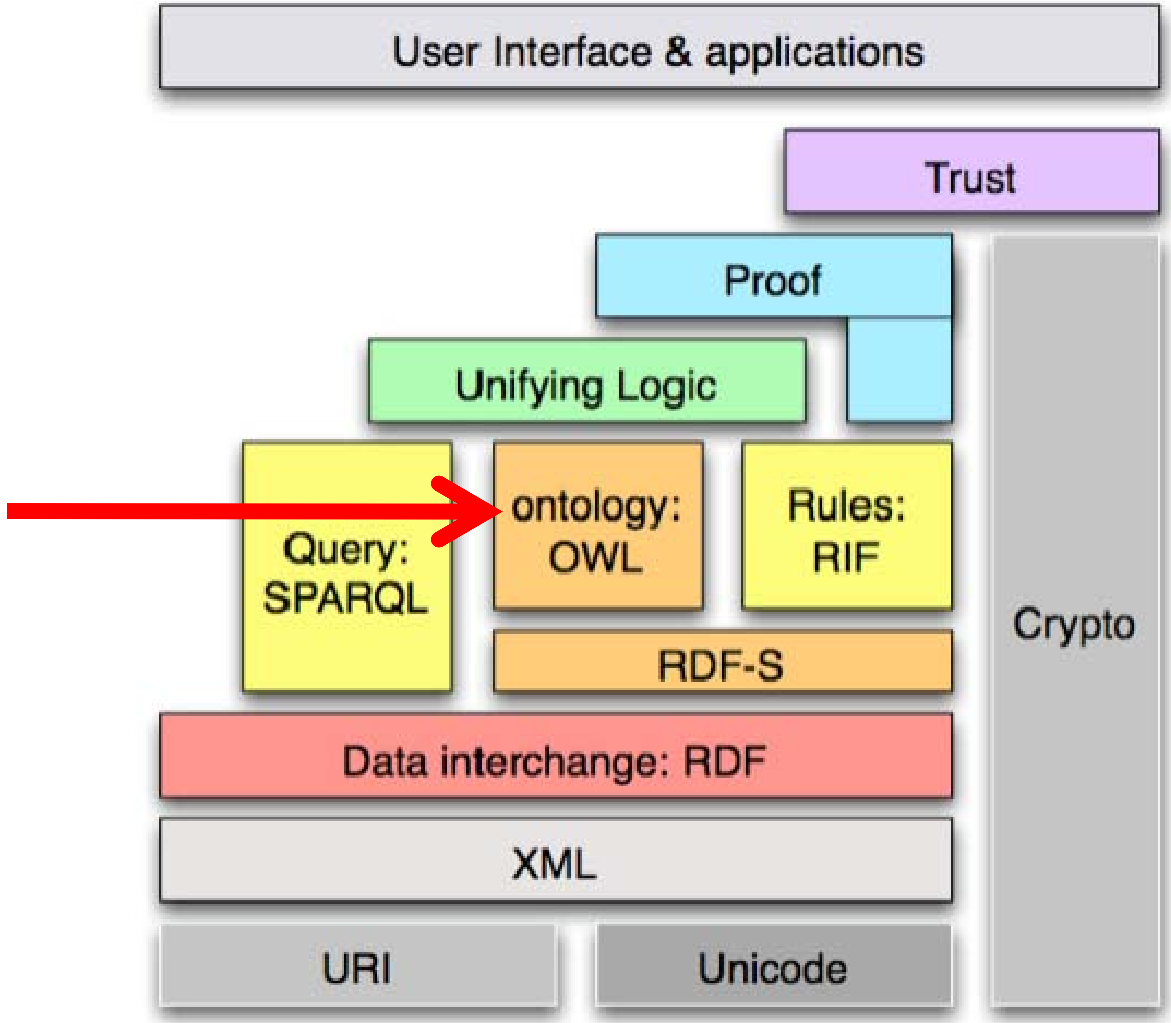
**Available from**

**http://www.semantic-web-book.org/page/ISWC2010_Tutorial**

# Part 1

# OWL 2 – Syntax, Semantics, Reasoning

# OWL


User Interface & applications
Trust
Proof
Unifying Logic
Query: SPARQL
ontology: OWL
Rules: RIF
RDF-S
Crypto
Data interchange: RDF
XML
URI
Unicode

# Main References Part 1


Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, Foundations of Semantic Web Technologies, Chapman & Hall/CRC, 2009

OWL 2 Document Overview: http://www.w3.org/TR/owl2-overview/

Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, Sebastian Rudolph, OWL 2 Web Ontology Language: Primer. W3C Recommendation, 27 October 2009. http://www.w3.org/TR/owl2-primer/

# OWL – Overview

- Web Ontology Language
  - W3C Recommendation for the Semantic Web, 2004
  - OWL 2 (revised W3C Recommendation), 2009

- Semantic Web KR language based on description logics (DLs)
  - OWL DL is essentially DL SROIQ(D)
  - KR for web resources, using URIs.
  - Using web-enabled syntaxes, e.g. based on XML or RDF. We present
    - DL syntax (used in research – not part of the W3C recommendation)
    - (some) RDF Turtle syntax

# Contents

# Contents

- OWL – Basic Ideas
- OWL as the Description Logic SROIQ(D)
- Different Perspectives on OWL
- OWL Semantics
- OWL Profiles
- Proof Theory
- Tools

# Rationale behind OWL

- Open World Assumption
- Favourable trade-off between expressivity and scalability
- Integrates with RDFS
- Purely declarative semantics

## Features:

- Fragment of first-order predicate logic (FOL)
- Decidable
- Known complexity classes (N2ExpTime for OWL 2 DL)
- Reasonably efficient for real KBs

# OWL Building Blocks

- individuals (written as URIs)
  - also: constants (FOL), resources (RDF)
  - http://example.org/sebastianRudolph
  - http://www.semantic-web-book.org
  - we write these lowercase and abbreviated, e.g. "sebastianRudolph"
- classes (also written as URIs!)
  - also: concepts, unary predicates (FOL)
  - we write these uppercase, e.g. "Father"
- properties (also written as URIs!)
  - also: roles (DL), binary predicates (FOL)
  - we write these lowercase, e.g. "hasDaughter"

# DL syntax FOL syntax

ABox statements

- Person(mary)
- Person(mary)

- Woman $\sqsubseteq$ Person
  - Person ≡ HumanBeing
- $\forall x$ (Woman(x) $\rightarrow$ Person(x))

- hasWife(john,mary)
- hasWife(john,mary)

- hasWife $\sqsubseteq$ hasSpouse
  - hasSpouse ≡ marriedWith
- $\forall x \forall y$ (hasWife(x,y)$\rightarrow$ hasSpouse(x,y))

TBox statements

# DL syntax FOL syntax

- Person(mary)
- Woman ⊑ Person
  - Person ≡ HumanBeing
- hasWife(john,mary)
- hasWife ⊑ hasSpouse
  - hasSpouse ≡ marriedWith

- :mary rdf:type :Person .
- :Woman rdfs:subClassOf :Person .
- :john :hasWife :mary .
- :hasWife rdfs:subPropertyOf :hasSpouse .

# Special classes and properties

KNO.E.SIS

- owl:Thing (RDF syntax)
  - DL-syntax: $\top$
  - contains everything
- owl:Nothing (RDF syntax)
  - DL-syntax: $\bot$
  - empty class
- owl:topProperty (RDF syntax)
  - DL-syntax: U
  - every pair is in U
- owl:bottomProperty (RDF syntax)
  - empty property

# Class constructors

- **conjunction**
  - **Mother ≡ Woman ⊓ Parent**
  - **:Mother owl:equivalentClass _:x .**
    **_:x rdf:type owl:Class .**
    **_:x owl:intersectionOf ( :Woman :Parent ) .**

∀x (Mother(x) ↔ Woman(x) ∧ Parent(x))

- **disjunction**
  - **Parent ≡ Mother ⊔ Father**
  - **:Parent owl:equivalentClass _:x .**
    **_:x rdf:type owl:Class .**
    **_:x owl:unionOf ( :Mother :Father ) .**

∀x (Parent(x) ↔ Mother(x) ∨ Father(x))

- **negation**
  - **ChildlessPerson ≡ Person ⊓ ¬Parent**
  - **:ChildlessPerson owl:equivalentClass _:x .**
    **_:x rdf:type owl:Class .**
    **_:x owl:intersectionOf ( :Person _:y ) .**
    **_:y owl:complementOf :Parent .**

∀x (ChildlessPerson(x) ↔ Person(x) ∧ ¬Parent(x))

# Class constructors

- **existential quantification**
  - **only to be used with a role – also called a *property restriction***
  - **Parent ≡ ∃hasChild.Person**
  - **:Parent owl:equivalentClass _:x .**
    **_:x rdf:type owl:Restriction .**
    **_:x owl:onProperty :hasChild .**
    **_:x owl:someValuesFrom :Person .**

$$\forall x\,(\mathrm{Parent}(x) \leftrightarrow \exists y\,(\mathrm{hasChild}(x,y) \wedge \mathrm{Person}(y)))$$

- **universal quantification**
  - **only to be used with a role – also called a *property restriction***
  - **Person ⊓ Happy ≡ ∀hasChild.Happy**
  - **_:x rdf:type owl:Class .**
    **_:x owl:intersectionOf ( :Person :Happy ) .**
    **_:x owl:equivalentClass _:y .**
    **_:y rdf:type owl:Restriction .**
    **_:y owl:onProperty :hasChild .**
    **_:y owl:allValuesFrom :Happy .**

$$\forall x\,(\mathrm{Person}(x) \wedge \mathrm{Happy}(x) \leftrightarrow \forall y\,(\mathrm{hasChild}(x,y) \rightarrow \mathrm{Happy}(y)))$$

- **Class constructors can be nested arbitrarily**

# Contents

- OWL – Basic Ideas
- OWL as the Description Logic SROIQ(D)
- Different Perspectives on OWL
- OWL Semantics
- OWL Profiles
- Proof Theory
- Tools

# Understanding SROIQ(D)

**The description logic ALC**

Complexity: ExpTime

- **ABox expressions:**
  **Individual assignments** **Father(john)**
  **Property assignments** **hasWife(john,mary)**

- **TBox expressions**
  **subclass relationships** $\sqsubseteq$

  **conjunction** $\sqcap$
  **disjunction** $\sqcup$
  **negation** $\neg$

  Also: $\top$, $\bot$

  **property restrictions** $\forall$
  $\exists$

# Understanding SROIQ(D)

**ALC + role chains = SR**

- **hasParent o hasBrother ⊑ hasUncle**

  $\forall x\ \forall y\ (\exists z\ ((hasParent(x,z) \wedge hasBrother(z,y)) \rightarrow hasUncle(x,y)))$

  - **includes top property and bottom property**

- **includes S = ALC + transitivity**
  - **hasAncestor o hasAncestor ⊑ hasAncestor**

- **includes SH = S + role hierarchies**
  - **hasFather ⊑ hasParent**

# Understanding SROIQ(D)

- O – nominals (closed classes)
  - MyBirthdayGuests $\equiv$ {bill,john,mary}
  - Note the difference to
    MyBirthdayGuests(bill)
    MyBirthdayGuests(john)
    MyBirthdayGuests(mary)

- Individual equality and inequality (no unique name assumption!)
  - bill = john
    - {bill} $\equiv$ {john}
  - bill $\neq$ john
    - {bill} $\sqcap$ {john} $\equiv$ $\bot$

# Understanding SROIQ(D)

- **I – inverse roles**
  - **hasParent $\equiv$ hasChild$^-$**
  - **Orphan $\equiv$ $\forall$hasChild$^-$.Dead**

- **Q – qualified cardinality restrictions**
  - **$\leq$4 hasChild.Parent(john)**
  - **HappyFather $\equiv$ $\geq$2 hasChild.Female**
  - **Car $\sqsubseteq$ =4hasTyre.$\top$**

- **Complexity SHIQ, SHOQ, SHIO: ExpTime.**
  **Complexity SHOIQ: NExpTime**
  **Complexity SROIQ: N2ExpTime**

# Understanding SROIQ(D)

Properties can be declared to be

| | |
|---|---|
| • Transitive | hasAncestor |
| • Symmetric | hasSpouse |
| • Asymmetric | hasChild |
| • Reflexive | hasRelative |
| • Irreflexive | parentOf |
| • Functional | hasHusband |
| • InverseFunctional | hasHusband |

called *property characteristics*

# Understanding SROIQ(D)

**(D) – datatypes**

- **so far, we have only seen properties with individuals in second argument, called *object properties* or *abstract roles* (DL)**
- **properties with datatype literals in second argument are called *data properties* or *concrete roles* (DL)**
- **allowed are many XML Schema datatypes, including xsd:integer, xsd:string, xsd:float, xsd:booelan, xsd:anyURI, xsd:dateTime**

  **and also e.g. owl:real**

# Understanding SROIQ(D)

**(D) – datatypes**

- **hasAge(john, "51"^^xsd:integer)**
- **additional use of *constraining facets* (from XML Schema)**
  - **e.g. Teenager ≡ Person ⊓ ∃hasAge.(xsd:integer: ≥12 and ≤19)**
    **note: this is not standard DL notation!**

# Understanding SROIQ(D)

**further expressive features**

- Self
  - PersonCommittingSuicide ≡ ∃kills.Self
- Keys (not really in SROIQ(D), but in OWL)
  - set of (object or data) properties whose values uniquely identify an object
- disjoint properties
  - Disjoint(hasParent,hasChild)
- explicit anonymous individuals
  - as in RDF: can be used instead of named individuals

# SROIQ(D) constructors – overview

- ABox assignments of individuals to classes or properties
- ALC: $\sqsubseteq$, $\equiv$ for classes
  $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$
  $\top$, $\bot$
- SR: + property chains, property characteristics,
  role hierarchies $\sqsubseteq$
- SRO: + nominals {o}
- SROI: + inverse properties
- SROIQ: + qualified cardinality constraints
- SROIQ(D): + datatypes (including facets)

- + top and bottom roles (for objects and datatypes)
- + disjoint properties
- + Self
- + Keys (not in SROIQ(D), but in OWL)

# Some Syntactic Sugar in OWL

**This applies to the non-DL syntaxes (e.g. RDF syntax).**

- **disjoint classes**
  - **Apple $\sqcap$ Pear $\sqsubseteq \bot$**
- **disjoint union**
  - **Parent $\equiv$ Mother $\sqcup$ Father**
    **Mother $\sqcap$ Father $\sqsubseteq \bot$**
- **negative property assignments (also for datatypes)**
  - **¬hasAge(jack,"53"^^xsd:integer)**

# Contents

# OWL – Extralogical Features

- OWL ontologies have URIs and can be referenced by others via
  - import statements
- Namespace declarations
- Entity declarations (must be done)
- Versioning information etc.

- Annotations
  - Entities and axioms (statements) can be endowed with annotations, e.g. using rdfs:comment.
  - OWL syntax provides *annotation properties* for this purpose.

# The modal logic perspective

- Description logics can be understood from a modal logic perspective.
- Each pair of $\forall R$ and $\exists R$ statements give rise to a pair of modalities.
- Essentially, some description logics are multi-modal logics.
- See e.g. Baader et al., The Description Logic Handbook, Cambridge University Press, 2007.

# The RDFS perspective

**RDFS semantics is weaker**

- :mary rdf:type :Person .
- :Mother rdfs:subClassOf :Woman .
- :john :hasWife :Mary .
- :hasWife rdfs:subPropertyOf :hasSpouse

- Person(mary)
- Mother $\sqsubseteq$ Woman
- hasWife(john,mary)
- hasWife $\sqsubseteq$ hasSpouse

- :hasWife rdfs:range :Woman .
- :hasWife rdfs:domain :Man .

- $\top \sqsubseteq \forall$hasWife.Woman
- $\top \sqsubseteq \forall$hasWife$^-$.Man or $\exists$hasWife.$\top \sqsubseteq$ Man

RDFS also allows to

- make statements about statements
  $\rightarrow$ only possible through annotations in OWL
- mix class names, individual names, property names (they are all URIs)
  $\rightarrow$ *punning* in OWL

# Punning

- **Description logics impose *type separation*, i.e. names of individuals, classes, and properties must be disjoint.**
- **In OWL 2 Full, type separation does not apply.**
- **In OWL 2 DL, type separation is relaxed, but a class X and an individual X are interpreted semantically as if they were different.**
- **Father(john)**
  **SocialRole(Father)**
- **See further below on the two different semantics for OWL.**

# Contents

# OWL Semantics

- There are two semantics for OWL.

1. Description Logic Semantics
   also: Direct Semantics; FOL Semantics
   Can be obtained by translation to FOL.
   Syntax restrictions apply! (see next slide)

2. RDF-based Semantics
   No syntax restrictions apply.
   Extends the direct semantics with RDFS-reasoning features.

In the following, we will deal with the direct semantics only.

# OWL Direct Semantics

**To obtain decidability, syntactic restrictions apply.**

- **Type separation / punning**
- **No cycles in property chains.**
- **No transitive properties in cardinality restrictions.**

# OWL Direct Semantics: Restrictions

- arbitrary property chain axioms lead to undecidability
- restriction: set of property chain axioms has to be *regular*
  - there must be a strict linear order $\prec$ on the properties
  - every property chain axiom has to have one of the following forms:

    $R \circ R \sqsubseteq R$ $\quad$ $S^- \sqsubseteq R$ $\quad$ $S_1 \circ S_2 \circ \ldots \circ S_n \sqsubseteq R$

    $R \circ S_1 \circ S_2 \circ \ldots \circ S_n \sqsubseteq R$ $\quad$ $S_1 \circ S_2 \circ \ldots \circ S_n \circ R \sqsubseteq R$
  - thereby, $S_i \prec R$ for all i= 1, 2, . . . , *n.*

- Example 1: $R \circ S \sqsubseteq R$ $\quad$ $S \circ S \sqsubseteq S$ $\quad$ $R \circ S \circ R \sqsubseteq T$

  → regular with order $S \prec R \prec T$
- Example 2: $R \circ T \circ S \sqsubseteq T$

  → not regular because form not admissible
- Example 3: $R \circ S \sqsubseteq S$ $\quad$ $S \circ R \sqsubseteq R$

  → not regular because no adequate order exists

# OWL Direct Semantics: Restrictions

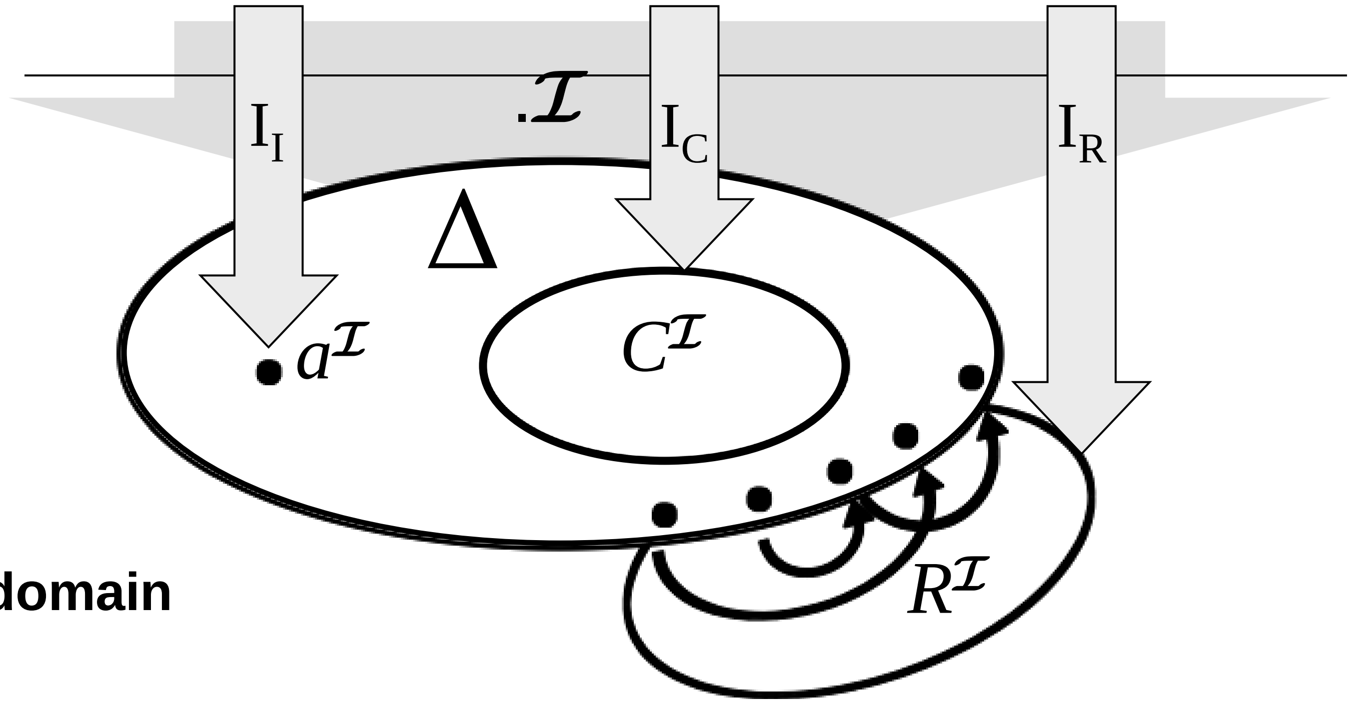
- combining property chain axioms and cardinality constraints may lead to undecidability
- restriction: use only *simple* properties in cardinality expressions (i.e. those which cannot be – directly or indirectly – inferred from property chains)
- technically:
  - for any property chain axiom $S_1 \circ S_2 \circ \ldots \circ S_n \sqsubseteq R$ with n>1, R is non-simple
  - for any subproperty axiom $S \sqsubseteq R$ with S non-simple, R is non-simple
  - all other properties are simple
- Example: $Q \circ P \sqsubseteq R$ $R \circ P \sqsubseteq R$ $R \sqsubseteq S$ $P \sqsubseteq R$ $Q \sqsubseteq S$

  non-simple: R, S simple: P, Q

# OWL Direct Semantics

- **model-theoretic semantics**
- **starts with interpretations**
- **an interpretation $\mathcal{I}$ maps**

  **individual names, class names and property names...**




**...into a domain**

# Interpretation Example

If we consider, for example, the knowledge base consisting of the axioms

$$\texttt{Professor} \sqsubseteq \texttt{FacultyMember}$$
$$\texttt{Professor}(\texttt{rudiStuder})$$
$$\texttt{hasAffiliation}(\texttt{rudiStuder}, \texttt{aifb})$$

then we could set

$$\begin{aligned}
\Delta &= \{a, b, \mathrm{Ian}\} \\
\mathrm{I}_{\mathbf{I}}(\texttt{rudiStuder}) &= \mathrm{Ian} \\
\mathrm{I}_{\mathbf{I}}(\texttt{aifb}) &= b \\
\mathrm{I}_{\mathbf{C}}(\texttt{Professor}) &= \{a\} \\
\mathrm{I}_{\mathbf{C}}(\texttt{FacultyMember}) &= \{a, b\} \\
\mathrm{I}_{\mathbf{R}}(\texttt{hasAffiliation}) &= \{(a, b), (b, \mathrm{Ian})\}
\end{aligned}$$

Intuitively, these settings are nonsense, but they nevertheless determine a valid interpretation.

# OWL Direct Semantics

- mapping is extended to complex class expressions:
  - $\top^I = \Delta^I$ $\bot^I = \emptyset$
  - $(C \sqcap D)^I = C^I \cap D^I$ $(C \sqcup D)^I = C^I \cup D^I$ $(\neg C)^I = \Delta^I \setminus C^I$
  - $(\forall R.C)^I = \{ x \mid \text{for all } (x,y) \in R^I \text{ we have } y \in C^I \}$
    $(\exists R.C)^I = \{ x \mid \text{there is } (x,y) \in R^I \text{ with } y \in C^I \}$
  - $(\geq nR.C)^I = \{ x \mid \#\{ y \mid (x,y) \in R^I \text{ and } y \in C^I \} \geq n \}$
  - $(\leq nR.C)^I = \{ x \mid \#\{ y \mid (x,y) \in R^I \text{ and } y \in C^I \} \leq n \}$
- ...and to role expressions:
  - $U^I = \Delta^I \times \Delta^I$ $(R^-)^I = \{ (y,x) \mid (x,y) \in R^I \}$
- ...and to axioms:
  - $C(a)$ holds, if $a^I \in C^I$ $R(a,b)$ holds, if $(a^I,b^I) \in R^I$
  - $C \sqsubseteq D$ holds, if $C^I \subseteq D^I$ $R \sqsubseteq S$ holds, if $R^I \subseteq S^I$
  - Disjoint(R,S) holds if $R^I \cap S^I = \emptyset$
  - $S_1 \circ S_2 \circ _{...} \circ S_n \sqsubseteq R$ holds if $S_1^I \circ S_2^I \circ _{...} \circ S_n^I \subseteq R^I$

# OWL Direct Semantics

- **what's below gives us a notion of *model*:**

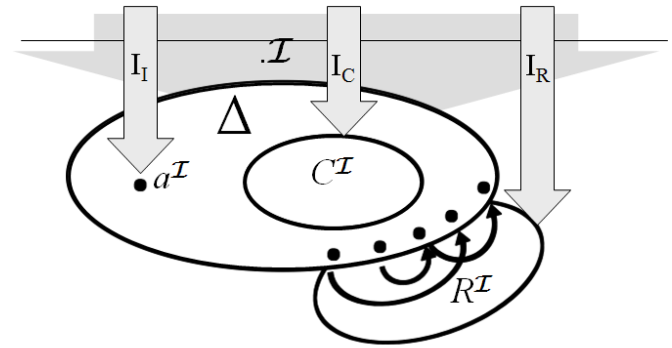
  **An interpretation is a model of a set of axioms if all the axioms hold (are evaluated to true) in the interpretation.**

- **Notion of *logical consequence* obtained via models (below).**

- **...and to axioms:**
  - **C(a) holds, if $a^I \in C^I$ R(a,b) holds, if $(a^I,b^I) \in R^I$**
  - **$C \sqsubseteq D$ holds, if $C^I \subseteq D^I$ $R \sqsubseteq S$ holds, if $R^I \subseteq S^I$**
  - **Disjoint(R,S) holds if $R^I \cap S^I = \emptyset$**
  - **$S_1 \circ S_2 \circ_{\dots} \circ S_n \sqsubseteq R$ holds if $S_1^I \circ S_2^I \circ_{\dots} \circ S_n^I \subseteq R^I$**

# Logical Consequence

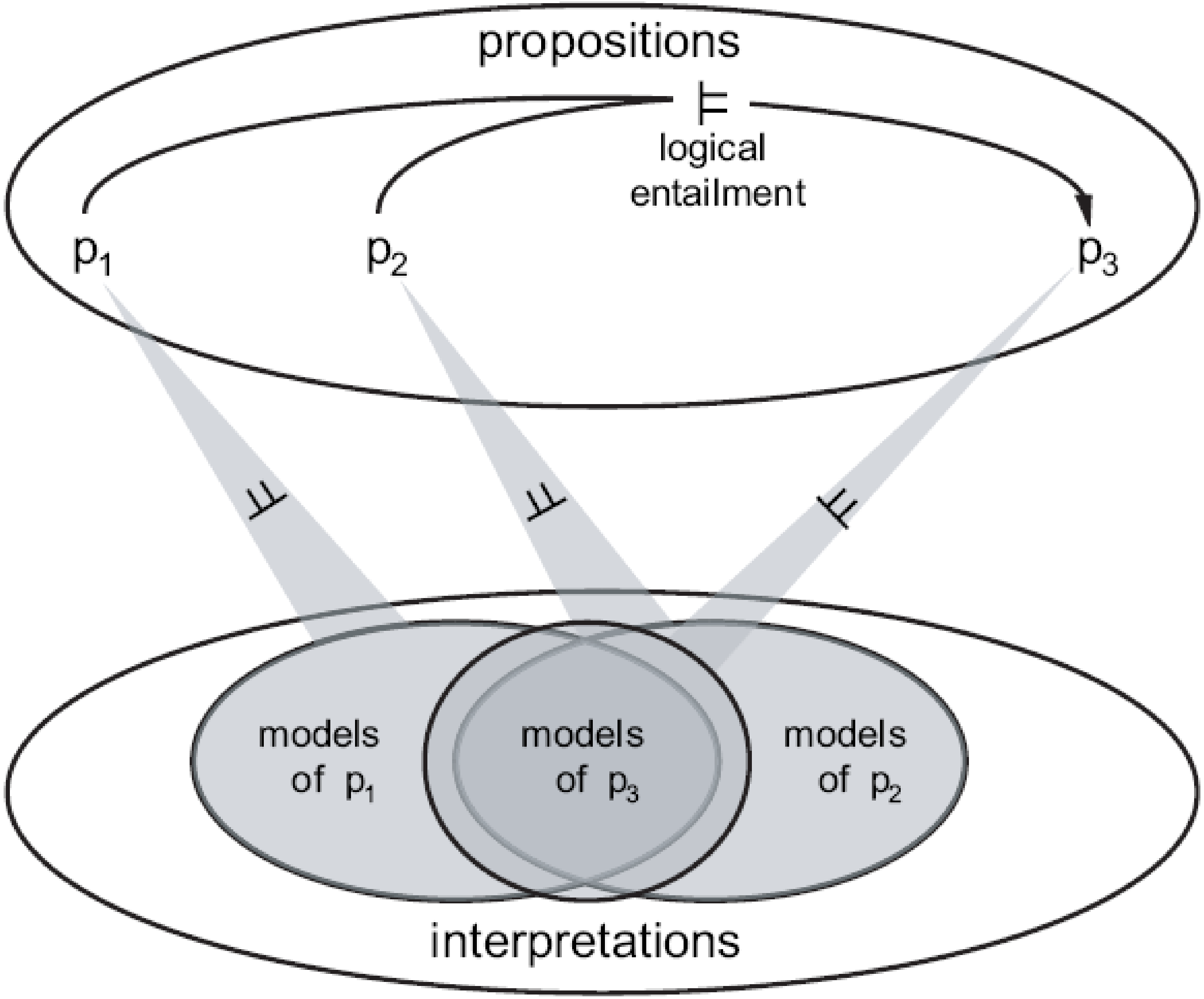
A *model* for an OWL KB is such a mapping I which satisfies all axioms in the KB.




An axiom $\alpha$ is a *logical consequence* of a KB if every model of the KB is also a model of $\alpha$.

The logical consequences of a KB are all those things which are *necessarily the case in all „realities“* in which the KB is the case.

# Notion of logical consequence



# Not a model!

If we consider, for example, the knowledge base consisting of the axioms

$$\texttt{Professor} \sqsubseteq \texttt{FacultyMember}$$
$$\texttt{Professor}(\texttt{rudiStuder})$$
$$\texttt{hasAffiliation}(\texttt{rudiStuder}, \texttt{aifb})$$

then we could set

$$\begin{aligned}
\Delta &= \{a, b, \mathrm{Ian}\} \\
\mathrm{I_I}(\texttt{rudiStuder}) &= \mathrm{Ian} \\
\mathrm{I_I}(\texttt{aifb}) &= b \\
\mathrm{I_C}(\texttt{Professor}) &= \{a\} \\
\mathrm{I_C}(\texttt{FacultyMember}) &= \{a, b\} \\
\mathrm{I_R}(\texttt{hasAffiliation}) &= \{(a, b), (b, \mathrm{Ian})\}
\end{aligned}$$

Intuitively, these settings are nonsense, but they nevertheless determine a valid interpretation.

# A model

$$\mathtt{Professor} \sqsubseteq \mathtt{FacultyMember}$$
$$\mathtt{Professor}(\mathtt{rudiStuder})$$
$$\mathtt{hasAffiliation}(\mathtt{rudiStuder}, \mathtt{aifb})$$

$$\Delta = \{a, r, s\}$$
$$\mathrm{I_I}(\mathtt{rudiStuder}) = r$$
$$\mathrm{I_I}(\mathtt{aifb}) = a$$
$$\mathrm{I_C}(\mathtt{Professor}) = \{r\}$$
$$\mathrm{I_C}(\mathtt{FacultyMember}) = \{r, s\}$$
$$\mathrm{I_R}(\mathtt{hasAffiliation}) = \{(r, a)\}$$

# Models

$$\mathtt{Professor} \sqsubseteq \mathtt{FacultyMember}$$

$$\mathtt{Professor}(\mathtt{rudiStuder})$$

$$\mathtt{hasAffiliation}(\mathtt{rudiStuder}, \mathtt{aifb})$$

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| $\Delta$ | $\{a, r, s\}$ | $\{1, 2\}$ | $\{\spadesuit\}$ |
| $I_I(\mathtt{rudiStuder})$ | $r$ | $1$ | $\spadesuit$ |
| $I_I(\mathtt{aifb})$ | $a$ | $2$ | $\spadesuit$ |
| $I_C(\mathtt{Professor})$ | $\{r\}$ | $\{1\}$ | $\{\spadesuit\}$ |
| $I_C(\mathtt{FacultyMember})$ | $\{a, r, s\}$ | $\{1, 2\}$ | $\{\spadesuit\}$ |
| $I_R(\mathtt{hasAffiliation})$ | $\{(r, a)\}$ | $\{(1, 1), (1, 2)\}$ | $\{(\spadesuit, \spadesuit)\}$ |

**Is FacultyMember(aifb) a logical consequence?**

# Counterexample

Returning to our running example knowledge base, let us show formally that $\texttt{FacultyMember}(\texttt{aifb})$ is not a logical consequence. This can be done by giving a model $M$ of the knowledge base where $\texttt{aifb}^M \notin \texttt{FacultyMember}^M$. The following determines such a model.

$$
\begin{aligned}
\Delta &= \{a, r\} \\
\mathrm{I}_{\mathbf{I}}(\texttt{rudiStuder}) &= r \\
\mathrm{I}_{\mathbf{I}}(\texttt{aifb}) &= a \\
\mathrm{I}_{\mathbf{C}}(\texttt{Professor}) &= \{r\} \\
\mathrm{I}_{\mathbf{C}}(\texttt{FacultyMember}) &= \{r\} \\
\mathrm{I}_{\mathbf{R}}(\texttt{hasAffiliation}) &= \{(r, a)\}
\end{aligned}
$$

# Logical Consequence

$$\mathtt{Professor} \sqsubseteq \mathtt{FacultyMember}$$

$$\mathtt{Professor(rudiStuder)}$$

$$\mathtt{hasAffiliation(rudiStuder, aifb)}$$

| | Model 1 | Model 2 | Model 3 |
|---|---|---|---|
| $\Delta$ | $\{a, r, s\}$ | $\{1, 2\}$ | $\{\spadesuit\}$ |
| $I_I(\mathtt{rudiStuder})$ | $r$ | $1$ | $\spadesuit$ |
| $I_I(\mathtt{aifb})$ | $a$ | $2$ | $\spadesuit$ |
| $I_C(\mathtt{Professor})$ | $\{r\}$ | $\{1\}$ | $\{\spadesuit\}$ |
| $I_C(\mathtt{FacultyMember})$ | $\{a, r, s\}$ | $\{1, 2\}$ | $\{\spadesuit\}$ |
| $I_R(\mathtt{hasAffiliation})$ | $\{(r, a)\}$ | $\{(1, 1), (1, 2)\}$ | $\{(\spadesuit, \spadesuit)\}$ |

**Is FacultyMember(rudiStuder) a logical consequence?**

# OWL Direct Semantics via FOL

- **but often OWL 2 DL is said to be a fragment of first-order predicate logic (FOL) [with equality]...**
- **yes, there is a translation of OWL 2 DL into FOL**

$$\pi(C \sqsubseteq D) = (\forall x)(\pi_x(C) \rightarrow \pi_x(D))$$
$$\pi_x(A) = A(x)$$
$$\pi_x(\neg C) = \neg\pi_x(C)$$
$$\pi_x(C \sqcap D) = \pi_x(C) \wedge \pi_x(D)$$
$$\pi_x(C \sqcup D) = \pi_x(C) \vee \pi_x(D)$$
$$\pi_x(\forall R.C) = (\forall x_1)(R(x, x_1) \rightarrow \pi_{x_1}(C))$$
$$\pi_x(\exists R.C) = (\exists x_1)(R(x, x_1) \wedge \pi_{x_1}(C))$$
$$\pi_x(\geq nS.C) = (\exists x_1)\ldots(\exists x_n)\left(\bigwedge_{i \neq j}(x_i \neq x_j) \wedge \bigwedge_i (S(x, x_i) \wedge \pi_{x_i}(C))\right)$$
$$\pi_x(\leq nS.C) = \neg(\exists x_1)\ldots(\exists x_{n+1})\left(\bigwedge_{i \neq j}(x_i \neq x_j) \wedge \bigwedge_i (S(x, x_i) \wedge \pi_{x_i}(C))\right)$$
$$\pi_x(\{a\}) = (x = a)$$
$$\pi_x(\exists S.\mathrm{Self}) = S(x, x)$$

$$\pi(R_1 \sqsubseteq R_2) = (\forall x)(\forall y)(\pi_{x,y}(R_1) \rightarrow \pi_{x,y}(R_2))$$
$$\pi_{x,y}(S) = S(x, y)$$
$$\pi_{x,y}(R^-) = \pi_{y,x}(R)$$
$$\pi_{x,y}(R_1 \circ \cdots \circ R_n) = (\exists x_1)\ldots(\exists x_{n-1})\left(\pi_{x,x_1}(R_1) \wedge \bigwedge_{i=1}^{n-2} \pi_{x_i,x_{i+1}}(R_{i+1}) \wedge \pi_{x_{n-1},y}(R_n)\right)$$
$$\pi(\mathrm{Ref}(R)) = (\forall x)\pi_{x,x}(R)$$
$$\pi(\mathrm{Asy}(R)) = (\forall x)(\forall y)(\pi_{x,y}(R) \rightarrow \neg\pi_{y,x}(R)$$
$$\pi(\mathrm{Dis}(R_1, R_2)) = \neg(\exists x)(\exists y)(\pi_{x,y}(R_1) \wedge \pi_{x,y}(R_2))$$

- **...which (interpreted under FOL semantics) coincides with the definition just given.**

# Inconsistency and Satisfiability

- **A set of axioms (knowledge base) is satisfiable (or consistent) if it has a model.**
- **It is unsatisfiable (inconsistent) if it does not have a model.**

- **Inconsistency is often caused by modeling errors.**

- **Unicorn(beauty)**
  **Unicorn $\sqsubseteq$ Fictitious**
  **Unicorn $\sqsubseteq$ Animal**
  **Animal $\sqsubseteq$ $\neg$Fictitious**

# Inconsistency and Satisfiability

- **A knowledge base is incoherent if a named class is equivalent to ⊥.**
- **It usually also points to a modeling error.**

$$\mathtt{Unicorn} \sqsubseteq \mathtt{Fictitious}$$
$$\mathtt{Unicorn} \sqsubseteq \mathtt{Animal}$$
$$\mathtt{Fictitious} \sqcap \mathtt{Animal} \sqsubseteq \bot$$

# A Semantic Puzzle

**From Horridge, Parsia, Sattler, From Justifications to Proofs for Entailments in OWL. In: Proceedings OWLED2009.**
**http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-529/**

$$\mathsf{Person} \sqsubseteq \neg\mathsf{Movie}$$

$$\mathsf{RRated} \sqsubseteq \mathsf{CatMovie}$$

$$\mathsf{CatMovie} \sqsubseteq \mathsf{Movie}$$

$$\mathsf{RRated} \equiv (\exists \mathsf{hasScript.ThrillerScript}) \sqcup (\forall \mathsf{hasViolenceLevel.High})$$

$$\mathsf{Domain}(\mathsf{hasViolenceLevel},\ \mathsf{Movie})$$

**Fig. 1.** A justification for $\mathsf{Person} \sqsubseteq \bot$

# What Semantics Is Good For

- Opinions Differ. Here's my take.

- Semantic Web requires a shareable, declarative and ***computable*** semantics.
- I.e., the semantics must be a formal entity which is clearly defined and automatically computable.

- Ontology languages provide this by means of their formal semantics.
- Semantic Web Semantics is given by a relation – the ***logical consequence*** **relation**.

- Note: This is considerably more than saying that the semantics of an ontology is the set of its logical consequences!

# In other words

We capture the meaning of information

not by specifying its meaning (which is impossible)
but by specifying

how information interacts with other information.

We describe the meaning indirectly through its effects.

# Simple Logical Reasoning

If I ask for soccer team members, I also want to get the goalkeepers listed ...

If I ask for cities, I also want all capitals listed ...

*inheritance reasoning*

# Less Simple Reasoning

What was again the name of that russian researcher who worked on resolution-based calculi for EL?

Are lobsters spiders?

What is "Käuzchen" in english?

*answering requires merging of knowledge from many websites and using background knowledge.*

# SNOMED CT

- **SNOMED CT: commercial ontology, medical domain ca. 300,000 axioms**

- **InjuryOfFinger $\equiv$ Injury $\sqcap$ $\exists$site.$Finger_S$**
  **InjuryOfHand $\equiv$ Injury $\sqcap$ $\exists$site.$Hand_S$**
  **$Finger_S \sqsubseteq Hand_P$**
  **$Hand_P \sqsubseteq Hand_S \sqcap \exists part.Hand_E$**

- **Reasoning has been used e.g. for**
  - **classification (computing the hidden taxonomy) e.g., InjuryOfFinger $\sqsubseteq$ InjuryOfHand**
  - **bug finding**

# Contents

- OWL – Basic Ideas
- OWL As the Description Logic SROIQ(D)
- Different Perspectives on OWL
- OWL Semantics
- OWL Profiles
- Proof Theory
- Tools

# OWL Profiles

- **OWL Full – using the RDFS-based semantics**
- **OWL DL – using the FOL semantics**

**The OWL 2 documents describe further profiles, which are of polynomial complexity:**

- **OWL EL (EL++)**
- **OWL QL (DL Lite$_R$)**
- **OWL RL (DLP)**

# OWL 2 EL

- **allowed:**
  - **subclass axioms with intersection, existential quantification, top, bottom**
    - **closed classs must have only one member**
  - **property chain axioms, range restrictions (under certain conditions)**
- **disallowed:**
  - **negation, disjunction, arbitrary universal quantification, role inverses**

$$\sqcap \exists \top \bot \sqsubseteq \sqcap \exists \top \bot$$

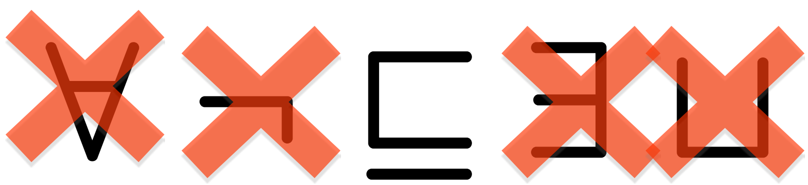
- **Examples:** $\text{Human} \sqsubseteq \exists \text{hasParent.Person}$
  $\exists \text{married}.\top \sqcap \text{CatholicPriest} \sqsubseteq \bot$;
  $\text{hasParent} \circ \text{hasParent} \sqsubseteq \text{hasGrandparent}$

# OWL 2 RL

- **Motivated by the question: what fraction of OWL 2 DL can be expressed naively by rules (with equality)?**
- **Examples:**
  - $\exists$parentOf.$\exists$parentOf.$\top \sqsubseteq$ Grandfather
    rule version: parentOf(x,y) parentOf(y,z) $\rightarrow$ Grandfather(x)
  - Orphan $\sqsubseteq \forall$hasParent.Dead
    rule version: Orphan(x) hasParent(x,y) $\rightarrow$ Dead(y)
  - Monogamous $\sqsubseteq \leq$1married.Alive
    rule version:
    Monogamous(x) married(x,y) Alive(y) married(x,z) Alive(z)$\rightarrow$ y=z
  - childOf $\circ$ childOf $\sqsubseteq$ grandchildOf
    rule version: childOf(x,y) childOf(y,z) $\rightarrow$ grandchildOf(x,z)
  - Disj(childOf,parentOf)
    rule version: childOf(x,y) parentOf(x,y) $\rightarrow$

# OWL 2 RL

- **Syntactic characterization:**
  - **essentially, all axiom types are allowed**
  - **disallow certain constructors on lhs and rhs of subclass statements**




  - **cardinality restrictions: only on rhs and only ≤1 and ≤0 allowed**
  - **closed classes: only with one member**

- **Reasoner conformance requires only soundness.**

# OWL 2 QL

- **Motivated by the question: what fraction of OWL 2 DL can be captured by standard database technology?**
- **Formally: query answering LOGSPACE w.r.t. data (via translation into SQL)**
- **Allowed:**
  - **subproperties, domain, range**
  - **subclass statements with**
    - **left hand side: class name or expression of type $\exists r.\top$**
    - **right hand side: intersection of class names, expressions of type $\exists r.C$ and negations of lhs expressions**
    - **no closed classes!**
- **Example:**

$$\exists \text{married}.\top \sqsubseteq \neg \text{Free} \sqcap \exists \text{has}.\text{Sorrow}$$

# Contents

- OWL – Basic Ideas
- OWL As the Description Logic SROIQ(D)
- Different Perspectives on OWL
- OWL Semantics
- OWL Profiles
- Proof Theory
- Tools

# A Reasoning Problem

**A is a logical consequence of K**

**written K ⊨ A**

**if and only if**

**every model of K is a model of A.**

- **To show an entailment, we need to check all models?**
- **But that‘s infinitely many!!!**

# A Reasoning Problem

We need algorithms which do not apply the model-based definition of logical consequence in a naive manner.

These algorithms should be syntax-based. (Computers can only do syntax manipulations.)

Luckily, such algorithms exist!

However, their correctness (soundness and completeness) needs to be proven formally.
Which is often a non-trivial problem requiring substantial mathematical build-up.

We won‘t do the proofs here.

# Proof Theory

We will show the Tableaux Method – implemented, e.g., in Pellet and Racer.

Alternatives:

- Transformation to disjunctive datalog using basic superposition done for SHIQ
- Naive mapping to Datalog for OWL RL
- Mapping to SQL for OWL QL
- Special-purpose algorithms for OWL EL e.g. transformation to Datalog

# Proof theory Via Tableaux

- Adaptation of FOL tableaux algorithms.
- Problem: OWL is decidable, but FOL tableaux algorithms do not guarantee termination.
- Solution: *blocking*.

# Contents

- Important inference problems
- Tableaux algorithm for ALC
- Tableaux algorithm for SHIQ

# Important Inference Problems

- Global consistency of a knowledge base. $KB \models$ **false**?
  - Is the knowledge base meaningful?
- Class consistency $C \equiv \bot$?
  - Is C necessarily empty?
- Class inclusion (Subsumption) $C \sqsubseteq D$?
  - Structuring knowledge bases
- Class equivalence $C \equiv D$?
  - Are two classes in fact the same class?
- Class disjointness $C \sqcap D = \bot$?
  - Do they have common members?
- Class membership $C(a)$?
  - Is a contained in C?
- Instance Retrieval „find all x with C(x)"
  - Find all (known!) individuals belonging to a given class.

# Reduction to Unsatisfiability

- Global consistency of a knowledge base. KB unsatisfiable
  - Failure to find a model.
- Class consistency $C \equiv \bot$?
  - KB $\cup$ {C(a)} unsatisfiable
- Class inclusion (Subsumption) $C \sqsubseteq D$?
  - KB $\cup$ {C $\sqcap$ $\neg$D(a)} unsatisfiable (a new)
- Class equivalence $C \equiv D$?
  - $C \sqsubseteq D$ und $D \sqsubseteq C$
- Class disjointness $C \sqcap D = \bot$?
  - KB $\cup$ {(C $\sqcap$ D)(a)} unsatisfiable (a new)
- Class membership C(a)?
  - KB $\cup$ {$\neg$C(a)} unsatisfiable
- Instance Retrieval „find all x with C(x)“
  - Check class membership for all individuals.

# Reduction to Satisfiability

- We will present so-called tableaux algorithms.
- They attempt to construct a model of the knowledge base in a „general, abstract“ manner.
  - If the construction fails, then (provably) there is no model – i.e. the knowledge base is unsatisfiable.
  - If the construction works, then it is satisfiable.

$\rightarrow$ Hence the reduction of all inference problems to the checking of unsatisfiability of the knowledge base!

# Contents

- Important inference problems
- Tableaux algorithm for ALC
- Tableaux algorithm for SHIQ

# ALC tableaux: contents

- Transformation to negation normal form
- Naive tableaux algorithm
- Tableaux algorithm with blocking

# Transform. to negation normal form

Given a knowledge base K.

- Replace $C \equiv D$ by $C \sqsubseteq D$ and $D \sqsubseteq C$.
- Replace $C \sqsubseteq D$ by $\neg C \sqcup D$.
- Apply the equations on the next slide exhaustively.

Resulting knowledge base: NNF(K)

*Negation normal form* of K.

Negation occurs only directly in front of atomic classes.

$$\begin{aligned}
\mathrm{NNF}(C) &= C \qquad \text{if } C \text{ is a class name}\\
\mathrm{NNF}(\neg C) &= \neg C \qquad \text{if } C \text{ is a class name}\\
\mathrm{NNF}(\neg\neg C) &= \mathrm{NNF}(C)\\
\mathrm{NNF}(C \sqcup D) &= \mathrm{NNF}(C) \sqcup \mathrm{NNF}(D)\\
\mathrm{NNF}(C \sqcap D) &= \mathrm{NNF}(C) \sqcap \mathrm{NNF}(D)\\
\mathrm{NNF}(\neg(C \sqcup D)) &= \mathrm{NNF}(\neg C) \sqcap \mathrm{NNF}(\neg D)\\
\mathrm{NNF}(\neg(C \sqcap D)) &= \mathrm{NNF}(\neg C) \sqcup \mathrm{NNF}(\neg D)\\
\mathrm{NNF}(\forall R.C) &= \forall R.\mathrm{NNF}(C)\\
\mathrm{NNF}(\exists R.C) &= \exists R.\mathrm{NNF}(C)\\
\mathrm{NNF}(\neg\forall R.C) &= \exists R.\mathrm{NNF}(\neg C)\\
\mathrm{NNF}(\neg\exists R.C) &= \forall R.\mathrm{NNF}(\neg C)
\end{aligned}$$

**K and NNF(K) have the same models (are *logically equivalent*).**

# Example

**$P \sqsubseteq (E \sqcap U) \sqcup \neg(\neg E \sqcup D)$.**

**In negation normal form:**

**$\neg P \sqcup (E \sqcap U) \sqcup (E \sqcap \neg D)$.**

# ALC tableaux: contents

- Transformation to negation normal form
- Naive tableaux algorithm
- Tableaux algorithm with blocking

# Naive tableaux algorithm

**Reduction to (un)satisfiability.**

**Idea:**

- **Given knowledge base K**
- **Attempt construction of a tree (called *Tableau*), which represents a model of K.**
  **(It's actually rather a *Forest*.)**
- **If attempt fails, K is unsatisfiable.**

# The Tableau

- **Nodes represent elements of the domain of the model**
  **$\rightarrow$ Every node x is labeled with a set L(x) of class expressions.**
  **C $\in$ L(x) means: "x is in the extension of C"**

- **Edges stand for role relationships:**
  **$\rightarrow$ Every edge <x,y> is labeled with a set L(<x,y>) of role names.**
  **R $\in$ L(<x,y>) means: "(x,y) is in the extension of R"**

# Simple example

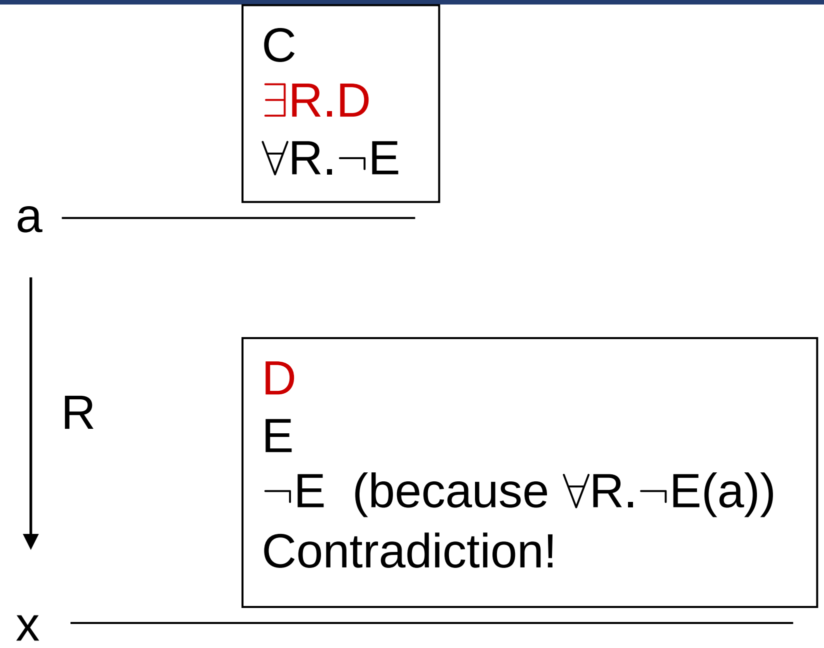
**C(a)**

**C ⊑ ∃R.D**

**D ⊑ E**

**Does this entail (∃R.E)(a)?**

**(add ∀R.¬E(a) and show unsatisfiability)**



# Another example

**C(a)**

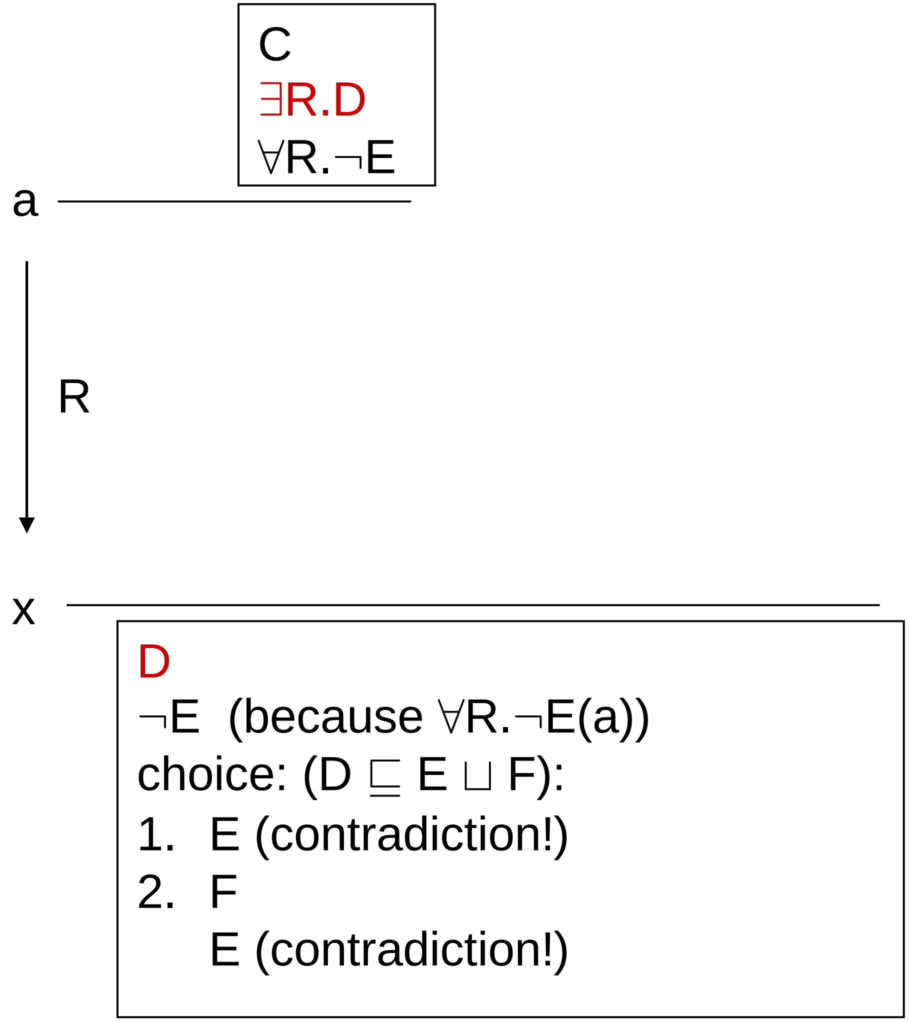
**$C \sqsubseteq \exists R.D$**

**$D \sqsubseteq E \sqcup F$**

**$F \sqsubseteq E$**

**Does this entail $(\exists R.E)(a)$?**

**(add $\forall R.\neg E(a)$ and show unsatisfiability)**



# Formal Definition

- Input: K=TBox + ABox (in NNF)
- Output: Whether or not K is satisfiable.

- A tableau is a directed labeled graph
  - nodes are individuals or (new) variable names
  - nodes x are labeled with sets L(x) of classes
  - edges <x,y> are labeled with sets L(<x,y>) of role names

# Initialisation

- Make a node for every individual in the ABox.
- Every node is labeled with the corresponding class names from the ABox.
- There is an edge, labeled with R, between a and b, if R(a,b) is in the ABox.

- (If there is no ABox, the initial tableau consists of a node x with empty label.)

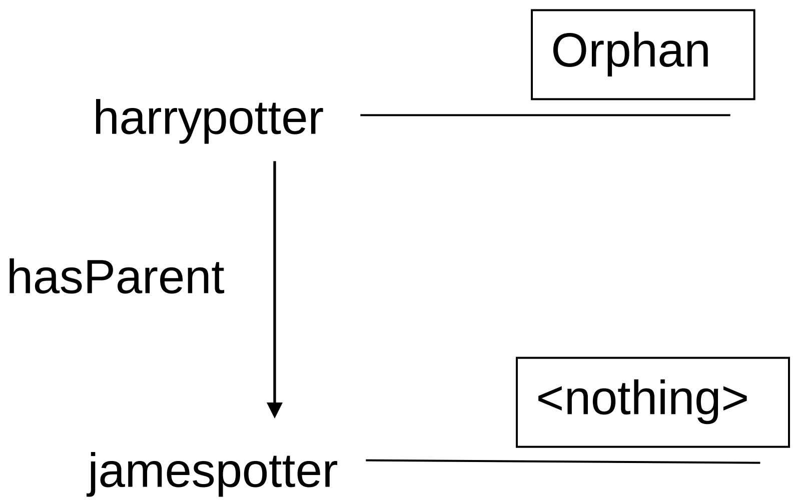
# Example initialisation

$\textbf{Human} \sqsubseteq \exists\textbf{hasParent.Human}$

$\textbf{Orphan} \sqsubseteq \textbf{Human} \sqcap \neg\exists\textbf{hasParent.Alive}$

**Orphan(harrypotter)**

**hasParent(harrypotter,jamespotter)**



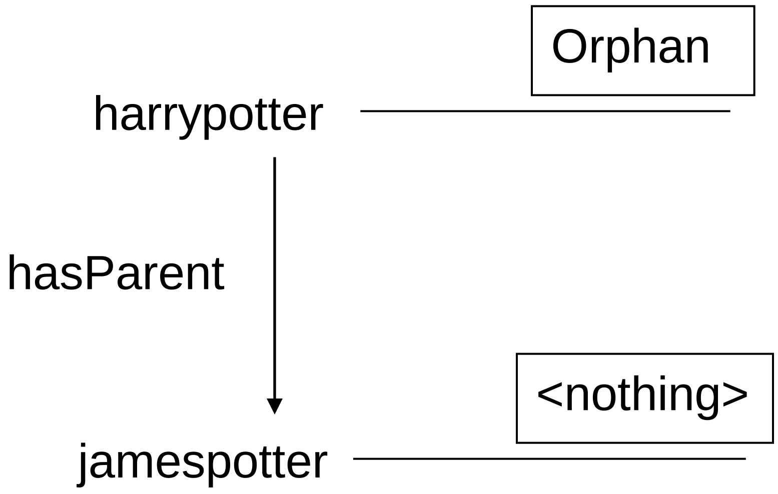
# Careful: need NNF!

**¬Human ⊔ ∃hasParent.Human**

**¬Orphan ⊔ (Human ⊓ ∀hasParent.¬Alive)**

**Orphan(harrypotter)**

**hasParent(harrypotter,jamespotter)**



# Constructing the tableau

- Non-deterministically extend the tableau using the rules on the next slide.

- Terminate, if
  - there is a contradiction in a node label (i.e., it contains classes C and ¬C, or it contains ⊥), or
  - none of the rules is applicable.

- If the tableau does not contain a contradiction, then the knowledge base is satisfiable.
  Or more precisely: If you can make a choice of rule applications such that no contradiction occurs and the process terminates, then the knowledge base is satisfiable.

# Naive ALC tableaux rules

**$\sqcap$-rule:** If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

**$\sqcup$-rule:** If $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

**$\exists$-rule:** If $\exists R.C \in \mathcal{L}(x)$ and there is no $y$ with $R \in L(x, y)$ and $C \in \mathcal{L}(y)$, then

1. add a new node with label $y$ (where $y$ is a new node label),
2. set $\mathcal{L}(x, y) = \{R\}$, and
3. set $\mathcal{L}(y) = \{C\}$.

**$\forall$-rule:** If $\forall R.C \in \mathcal{L}(x)$ and there is a node $y$ with $R \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

**TBox-rule:** If $C$ is a TBox statement and $C \notin \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$.

# Example

$\neg$Alive(jamespotter)
i.e. add: Alive(jamespotter)
and search for contradiction

**$\neg$Human $\sqcup$ $\exists$hasParent.Human**

**$\neg$Orphan $\sqcup$ (Human $\sqcap$ $\forall$hasParent.$\neg$Alive)**

**Orphan(harrypotter)**

**hasParent(harrypotter,jamespotter)**

harrypotter

Orphan
$\neg$Orphan $\sqcup$ (Human $\sqcap$ $\forall$hasParent.$\neg$Alive)
1. $\neg$Orphan (contradiction)
2. Human $\sqcap$ $\forall$hasParent.$\neg$Alive
   Human
   $\forall$hasParent.$\neg$Alive

hasParent

Alive
2. $\neg$Alive (contradiction)

jamespotter

# ALC tableaux: contents

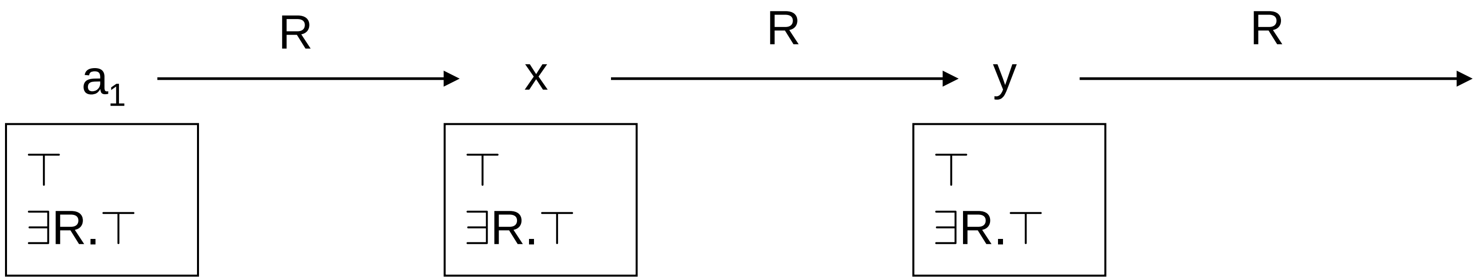
- Transformation to negation normal form
- Naive tableaux algorithm
- Tableaux algorithm with blocking

# There's a termination problem

**TBox: $\exists R.\top$**

**ABox: $\top(a_1)$**

- **Obviously satisfiable:
Model M with domain elements $a_1^M, a_2^M, \ldots$
and $R^M(a_i^M, a_{i+1}^M)$ for all $i \geq 1$**
- **but tableaux algorithm does not terminate!**

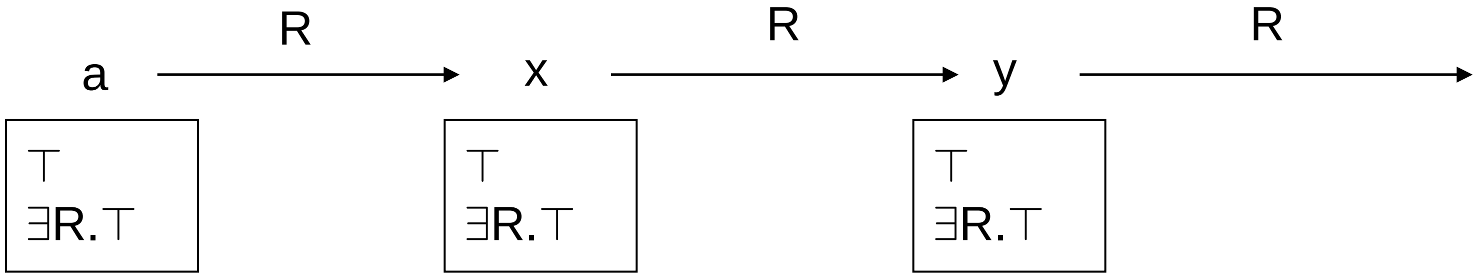


# Solution?

**Actually, things repeat!**

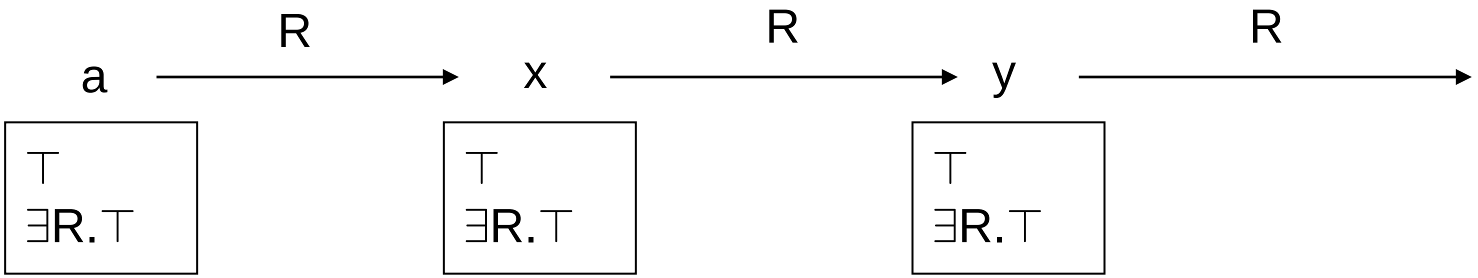
**Idea: it is not necessary to expand x, since it‘s simply a copy of a.**

**⇒ Blocking**



# Blocking

- x is *blocked* (by y) if
  - x is not an individual (but a variable)
  - y is a predecessor of x and $L(x) \subseteq L(y)$
  - or a predecessor of x is blocked




**Here, x is blocked by a.**

# Constructing the tableau

- Non-deterministically extend the tableau using the rules on the next slide, but only apply a rule if x is not blocked!
- Terminate, if
  - there is a contradiction in a node label (i.e., it contains classes C and ¬C), or
  - none of the rules is applicable.
- If the tableau does not contain a contradiction, then the knowledge base is satisfiable.
  Or more precisely: If you can make a choice of rule applications such that no contradiction occurs and the process terminates, then the knowledge base is satisfiable.

# Naive ALC tableaux rules

**$\sqcap$-rule:** If $C \sqcap D \in \mathcal{L}(x)$ and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

**$\sqcup$-rule:** If $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

**$\exists$-rule:** If $\exists R.C \in \mathcal{L}(x)$ and there is no $y$ with $R \in L(x, y)$ and $C \in \mathcal{L}(y)$, then

1. add a new node with label $y$ (where $y$ is a new node label),
2. set $\mathcal{L}(x, y) = \{R\}$, and
3. set $\mathcal{L}(y) = \{C\}$.

**$\forall$-rule:** If $\forall R.C \in \mathcal{L}(x)$ and there is a node $y$ with $R \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

**TBox-rule:** If $C$ is a TBox statement and $C \notin \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$.

**Apply only if x is not blocked!**

# Example (0)

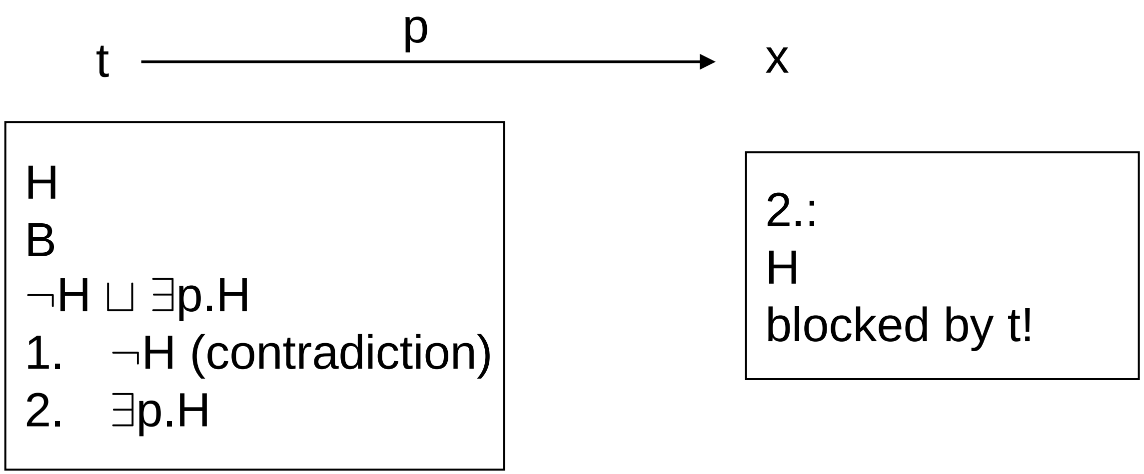
- **Knowledge base {Human $\sqsubseteq$ ∃hasParent.Human, Bird(tweety)}**
- **We want to show that Human(tweety) does *not* hold,
i.e. that ¬Human(tweety) is entailed.**
- **We will not be able to show this.
I.e. Human(tweety) is *possible*.**

- **Shorter notation:**
**H $\sqsubseteq$ ∃p.H**
**B(t)**

**¬H(t) entailed?**

# Example (0)

**Knowledge base {¬H ⊔ ∃p.H, B(t), H(t)}**




**expansion stops. Cannot find contradiction!**

# Example (0) the other case

**Knowledge base {¬H ⊔ ∃p.H, B(t), ¬H(t)}**

t —p→ x —p→ y

¬H
B
¬H ⊔ ∃p.H
1. ¬H cannot be added. no expansion in this part
2. ∃p.H

2.:
H
¬H ⊔ ∃p.H
2.1: ¬H (contradiction)
2.2: ∃p.H

2.2:
H
blocked by x

**no further expansion possible – knowledge base is satisfiable!**

# Example(1)

**Show, that**

**Professor ⊑ (Person ⊓ Unversitymember)**
**⊔ (Person ⊓ ¬PhDstudent)**

**entails that every Professor is a Person.**

Find contradiction in:
¬P ⊔ (E ⊓ U) ⊔ (E ⊓ ¬S)
P ⊓ ¬E(x)

x

P ⊓ ¬E
P
¬E
¬P ⊔ (E ⊓ U) ⊔ (E ⊓ ¬S)

1. ¬P (contradiction)
2. (E ⊓ U) ⊔ (E ⊓ ¬S)
    1. E ⊓ U
       E (contradiction)
    2. E ⊓ ¬S
       E (contradiction)

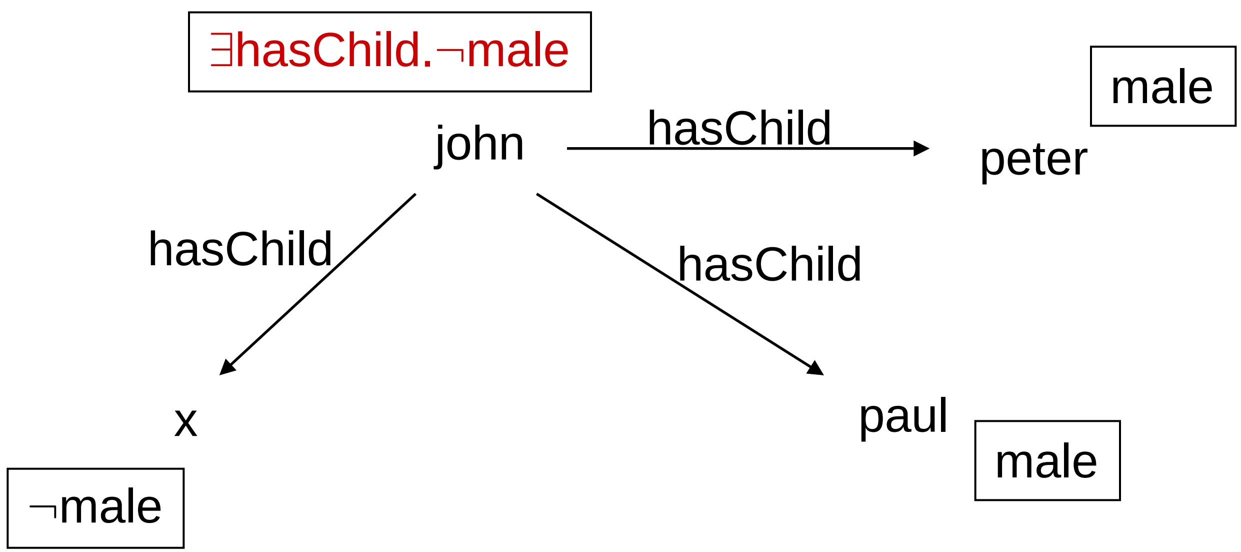
# Example (2)

**Show that**

**hasChild(john, peter)**
**hasChild(john, paul)**
**male(peter)**
**male(paul)**

**does *not* entail ∀hasChild.male(john).**

¬∀hasChild.male ≡ ∃hasChild.¬male



# Example (3)

**Show that the knowledge base**

**Bird $\sqsubseteq$ Flies**

**Penguin $\sqsubseteq$ Bird**

**Penguin $\sqcap$ Flies $\sqsubseteq \bot$**

**Penguin(tweety)**

**is unsatisfiable.**

TBox:
$\neg B \sqcup F$
$\neg P \sqcup B$
$\neg P \sqcup \neg F \sqcup \bot$

tweety

P
$\neg P \sqcup B$
$\neg B \sqcup F$
$\neg P \sqcup \neg F$

1. $\neg P$ (contradiction)
2. B
   1. $\neg B$ (contradiction)
   2. F
      1. $\neg P$ (contradiction)
      2. $\neg F$ (contradiction)

# Example (4)

**Show that the knowledge base**

**C(a)** **C(c)**

**R(a,b)** **R(a,c)**

**S(a,a)** **S(c,b)**

**$C \sqsubseteq \forall S.A$**

**$A \sqsubseteq \exists R.\exists S.A$**

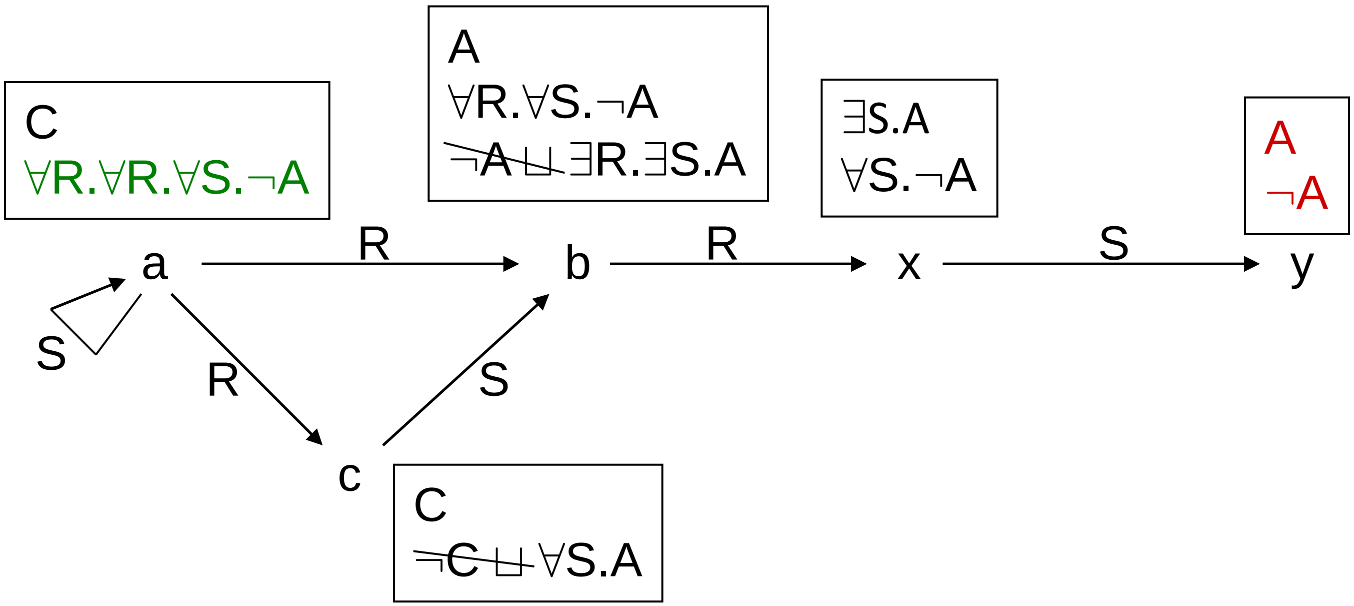
**$A \sqsubseteq \exists R.C$**

**entails $\exists R.\exists R.\exists S.A(a)$.**

# Example (4)

¬∃R.∃R.∃S.A ≡ ∀R.∀R.∀S.¬A

TBox:
¬C ⊔ ∀S.A
¬A ⊔ ∃R.∃S.A
¬A ⊔ ∃R.C



# Contents

- Important inference problems
- Tableaux algorithm for ALC
- Tableaux algorithm for SHIQ

# Tableaux Algorithm for SHIQ

- Basic idea is the same.
- Blocking rule is more complicated
- Other modifictions are also needed.

# Transform. to negation normal form

Given a knowledge base K.

- Replace $C \equiv D$ by $C \sqsubseteq D$ and $D \sqsubseteq C$.
- Replace $C \sqsubseteq D$ by $\neg C \sqcup D$.
- Apply the equations on the next slide exhaustively.

Resulting knowledge base: NNF(K)

*Negation normal form* of K.

Negation occurs only directly in front of atomic classes.

$$
\begin{aligned}
\mathrm{NNF}(C) &= C \quad \text{if } C \text{ is a class name} \\
\mathrm{NNF}(\neg C) &= \neg C \quad \text{if } C \text{ is a class name} \\
\mathrm{NNF}(\neg\neg C) &= \mathrm{NNF}(C) \\
\mathrm{NNF}(C \sqcup D) &= \mathrm{NNF}(C) \sqcup \mathrm{NNF}(D) \\
\mathrm{NNF}(C \sqcap D) &= \mathrm{NNF}(C) \sqcap \mathrm{NNF}(D) \\
\mathrm{NNF}(\neg(C \sqcup D)) &= \mathrm{NNF}(\neg C) \sqcap \mathrm{NNF}(\neg D) \\
\mathrm{NNF}(\neg(C \sqcap D)) &= \mathrm{NNF}(\neg C) \sqcup \mathrm{NNF}(\neg D) \\
\mathrm{NNF}(\forall R.C) &= \forall R.\mathrm{NNF}(C) \\
\mathrm{NNF}(\exists R.C) &= \exists R.\mathrm{NNF}(C) \\
\mathrm{NNF}(\neg\forall R.C) &= \exists R.\mathrm{NNF}(\neg C) \\
\mathrm{NNF}(\neg\exists R.C) &= \forall R.\mathrm{NNF}(\neg C)
\end{aligned}
$$

**NNF($\leq$n R.C) = $\leq$n R.NNF(C)**

**NNF($\geq$n R.C) = $\geq$n R.NNF(C)**

**NNF($\neg$ $\leq$n R.C) = $\geq$(n+1)R.NNF(C)**

**NNF($\neg$ $\geq$n R.C) = $\leq$(n-1)R.NNF(C), where $\leq$(-1)R.C = $\bot$**

**K and NNF(K) have the same models (are *logically equivalent*).**

# Formal Definition

- A tableau is a directed labeled graph
  - nodes are individuals or (new) variable names
  - nodes x are labeled with sets L(x) of classes
  - edges <x,y> are labeled
    - either with sets L(<x,y>) of role names or inverse role names
    - or with the symbol = (for equality)
    - or with the symbol ≠ (for inequality)

# Initialisation

- Make a node for every individual in the ABox. These nodes are called *root nodes*.
- Every node is labeled with the corresponding class names from the ABox.
- There is an edge, labeled with R, between a and b, if R(a,b) is in the ABox.
- There is an edge, labeled ≠, between a and b if a ≠ b is in the ABox.
- There are no = relations (yet).

# Notions

- We write $S^{--}$ as S.
- If R $\in$ L(<x,y>) and R $\sqsubseteq$ S (where R,S can be inverse roles), then
  - y is an S-successor of x and
  - x is an S-predecessor of y.
- If y is an S-successor or an $S^{-}$-predecessor of x, then y is an *neighbor* of x.
- *Ancestor* is the transitive closure of *Predecessor.*

# Blocking for SHIQ

- x is *blocked* by y if x,y are not root nodes and
  - the following hold: ["x is directly blocked"]
    - no ancestor of x is blocked
    - there are predecessors y', x' of x
    - y is a successor of y' and x is a successor of x'
    - L(x) = L(y) and L(x') = L(y')
    - L(<x',x>) = L(<y',y>)
  - or the following holds: ["x is indirectly blocked"]
    - an ancestor of x is blocked or
    - x is successor of some y with L(<y,x>) = ∅

# Constructing the tableau

- Non-deterministically extend the tableau using the rules on the next slide.

- Terminate, if
  – there is a contradiction in a node label, i.e.,
    - it contains $\bot$ or classes C and $\neg$C or
    - it contains a class $\leq$ nR.C and x also has (n+1) R-successors $y_i$ and $y_i \neq y_j$ (for all $i \neq j$)
  – or none of the rules is applicable.

- If the tableau does not contain a contradiction, then the knowledge base is satisfiable.
  Or more precisely: If you can make a choice of rule applications such that no contradiction occurs and the process terminates, then the knowledge base is satisfiable.

# SHIQ Tableaux Rules

**$\sqcap$-rule:** If $x$ is not indirectly blocked, $C \sqcap D \in \mathcal{L}(x)$, and $\{C, D\} \not\subseteq \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow \{C, D\}$.

**$\sqcup$-rule:** If $x$ is not indirectly blocked, $C \sqcup D \in \mathcal{L}(x)$ and $\{C, D\} \sqcap \mathcal{L}(x) = \emptyset$, then set $\mathcal{L}(x) \leftarrow C$ or $\mathcal{L}(x) \leftarrow D$.

$\exists$-rule: If $x$ is not blocked, $\exists R.C \in \mathcal{L}(x)$, and there is no $y$ with $R \in \mathcal{L}(x, y)$ and $C \in \mathcal{L}(y)$, then

1. add a new node with label $y$ (where $y$ is a new node label),
2. set $\mathcal{L}(x, y) = \{R\}$ and $\mathcal{L}(y) = \{C\}$.

**$\forall$-rule:** If $x$ is not indirectly blocked, $\forall R.C \in \mathcal{L}(x)$, and there is a node $y$ with $R \in \mathcal{L}(x, y)$ and $C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow C$.

**TBox-rule:** If $x$ is not indirectly blocked, $C$ is a TBox statement, and $C \notin \mathcal{L}(x)$, then set $\mathcal{L}(x) \leftarrow C$.

# SHIQ Tableaux Rules

**trans-rule:** If $x$ is not indirectly blocked, $\forall S.C \in \mathcal{L}(x)$, $S$ has a transitive subrole $R$, and $x$ has an $R$-neighbor $y$ with $\forall R.C \notin \mathcal{L}(y)$, then set $\mathcal{L}(y) \leftarrow \forall R.C$.

**choose-rule:** If $x$ is not indirectly blocked, $\leq nS.C \in \mathcal{L}(x)$ or $\geq nS.C \in \mathcal{L}(x)$, and there is an $S$-neighbor $y$ of $x$ with $\{C, \text{NNF}(\neg C)\} \cap \mathcal{L}(y) = \emptyset$, then set $\mathcal{L}(y) \leftarrow C$ or $\mathcal{L}(y) \leftarrow \text{NNF}(\neg C)$.

**$\geq$-rule:** If $x$ is not blocked, $\geq nS.C \in \mathcal{L}(x)$, and there are no $n$ $S$-neighbors $y_1, \ldots, y_n$ of $x$ with $C \in \mathcal{L}(y_i)$ and $y_i \not\approx y_j$ for $i, j \in \{1, \ldots, n\}$ and $i \neq j$, then

1. create $n$ new nodes with labels $y_1, \ldots, y_n$ (where the labels are new),
2. set $\mathcal{L}(x, y_i) = \{S\}$, $\mathcal{L}(y_i) = \{C\}$, and $y_i \not\approx y_j$ for all $i, j \in \{1, \ldots, n\}$ with $i \neq j$.

**$\leq$-rule:** If $x$ is not indirectly blocked, $\leq nS.C \in \mathcal{L}(x)$, there are more than $n$ $S$-neighbors $y_i$ of $x$ with $C \in \mathcal{L}(y_i)$, and $x$ has two $S$-neighbors $y, z$ such that $y$ is neither a root node nor an ancestor of $z$, $y \not\approx z$ does not hold, and $C \in \mathcal{L}(y) \cap \mathcal{L}(z)$, then

1. set $\mathcal{L}(z) \leftarrow \mathcal{L}(y)$,
2. if $z$ is an ancestor of $x$, then $\mathcal{L}(z, x) \leftarrow \{\text{Inv}(R) \mid R \in \mathcal{L}(x, y)\}$,
3. if $z$ is not an ancestor of $x$, then $\mathcal{L}(x, z) \leftarrow \mathcal{L}(x, y)$,
4. set $\mathcal{L}(x, y) = \emptyset$, and
5. set $u \not\approx z$ for all $u$ with $u \not\approx y$.

**$\leq$-root-rule:** If $\leq nS.C \in \mathcal{L}(x)$, there are more than $n$ $S$-neighbors $y_i$ of $x$ with $C \in \mathcal{L}(y_i)$, and $x$ has two $S$-neighbors $y, z$ which are both root nodes, $y \not\approx z$ does not hold, and $C \in \mathcal{L}(y) \cap \mathcal{L}(z)$, then

1. set $\mathcal{L}(z) \leftarrow \mathcal{L}(y)$,
2. for all directed edges from $y$ to some $w$, set $\mathcal{L}(z, w) \leftarrow \mathcal{L}(y, w)$,
3. for all directed edges from some $w$ to $y$, set $\mathcal{L}(w, z) \leftarrow \mathcal{L}(w, y)$,
4. set $\mathcal{L}(y) = \mathcal{L}(w, y) = \mathcal{L}(y, w) = \emptyset$ for all $w$,
5. set $u \not\approx z$ for all $u$ with $u \not\approx y$, and
6. set $y \approx z$.

# Example (1): cardinalities

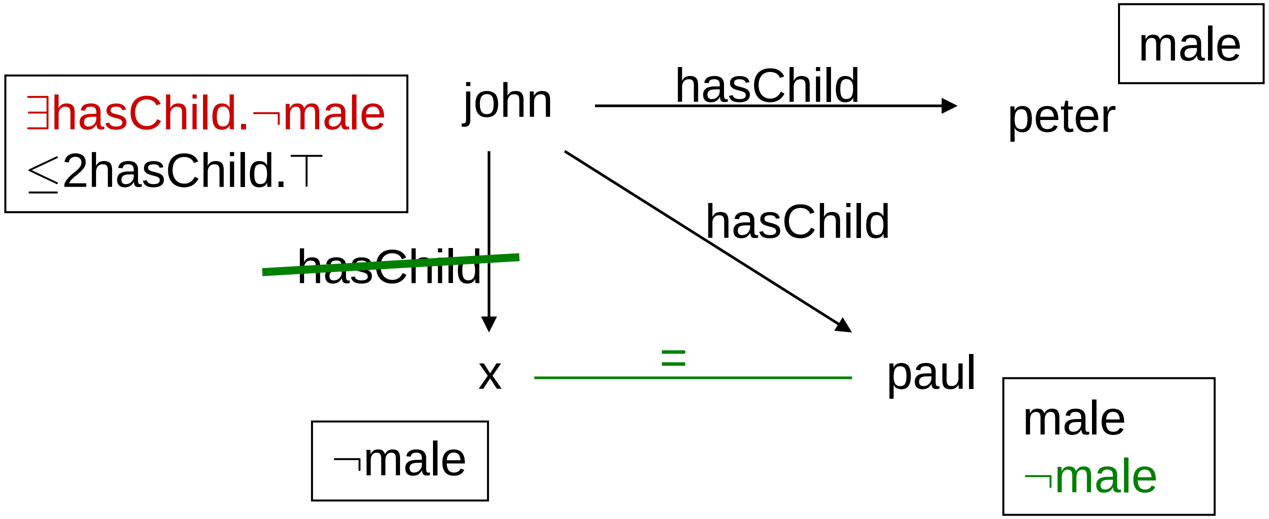
**Show, that**

**hasChild(john, peter)**
**hasChild(john, paul)**
**male(peter)**
**male(paul)**
**$\leq$2hasChild.$\top$(john)**

**does *not* entail $\forall$hasChild.male(john).**

$\neg\forall$hasChild.male $\equiv$ $\exists$hasChild.$\neg$male

now apply $\leq$



# Example (1): cardinalities

**Show, that**

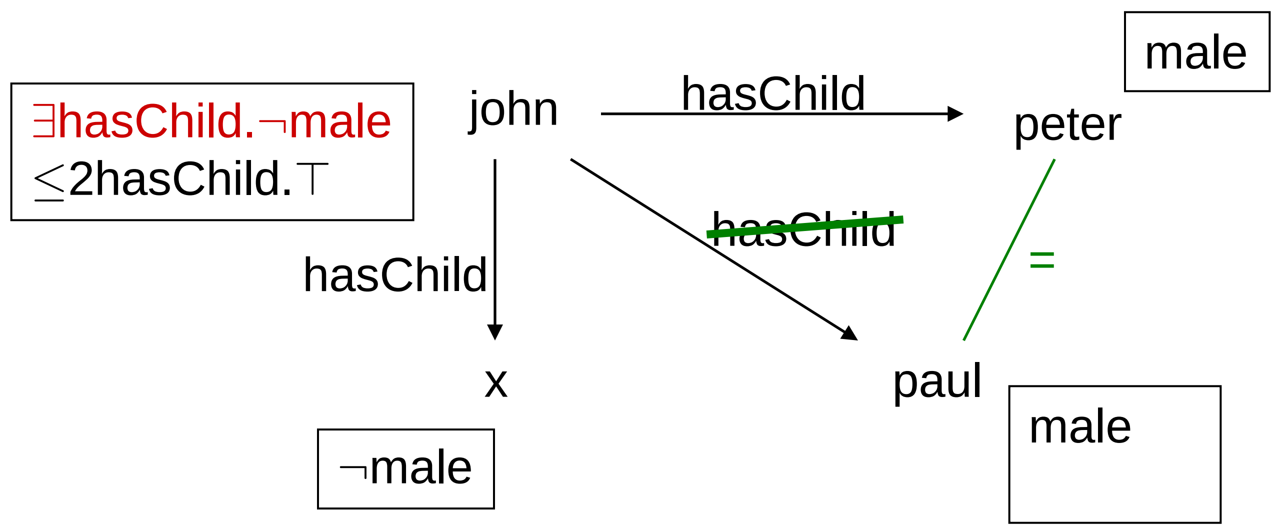
**hasChild(john, peter)**
**hasChild(john, paul)**
**male(peter)**
**male(paul)**
**$\leq$2hasChild.$\top$(john)**

**does *not* entail $\forall$hasChild.male(john).**

$\neg\forall$hasChild.male $\equiv$ $\exists$hasChild.$\neg$male

backtracking!

now apply $\leq$



# Example (1): cardinalities – again

**Show, that**

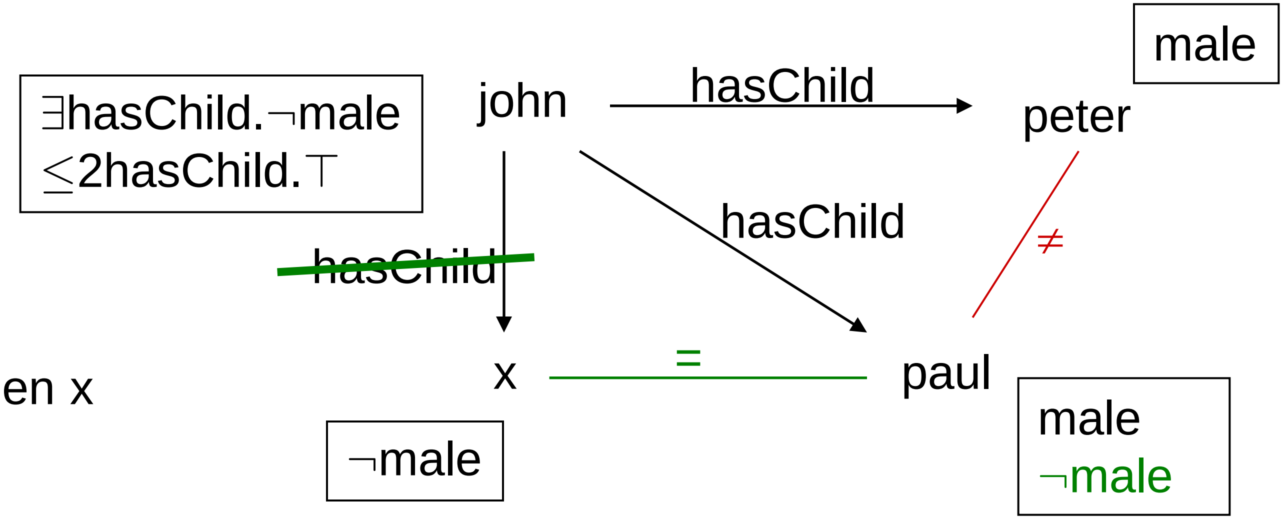
**hasChild(john, peter)**
**hasChild(john, paul)**
**male(peter)**
**male(paul)**
**≤2hasChild.⊤(john) and peter ≠ paul**

**does *not* entail ∀hasChild.male(john).**

¬∀hasChild.male ≡ ∃hasChild.¬male

now apply ≤

can backtrack only between x and peter – also leads to contradiction



# Example (2): cardinalities

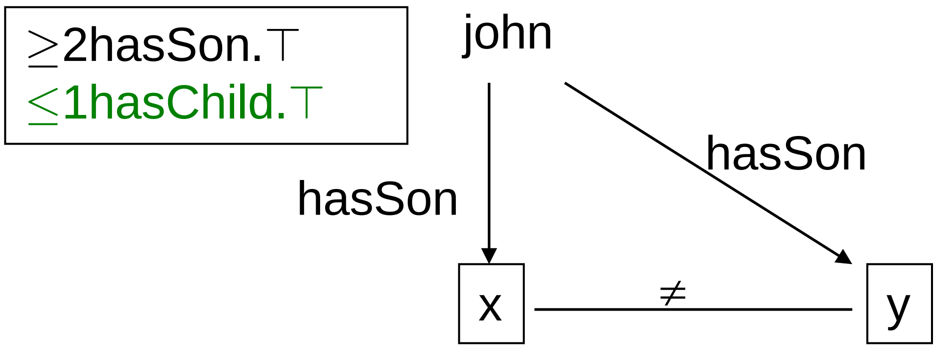
**Show, that**

**$\geq 2 hasSon.\top(john)$**

**entails $\geq 2 hasChild.\top(john)$.**

$\neg\geq 2 hasSon.\top \equiv \leq 1 hasChild.\top$

**$hasSon \sqsubseteq hasChild$**




hasSon-neighbors are also hasChild-neighbors, tableau terminates with contradiction

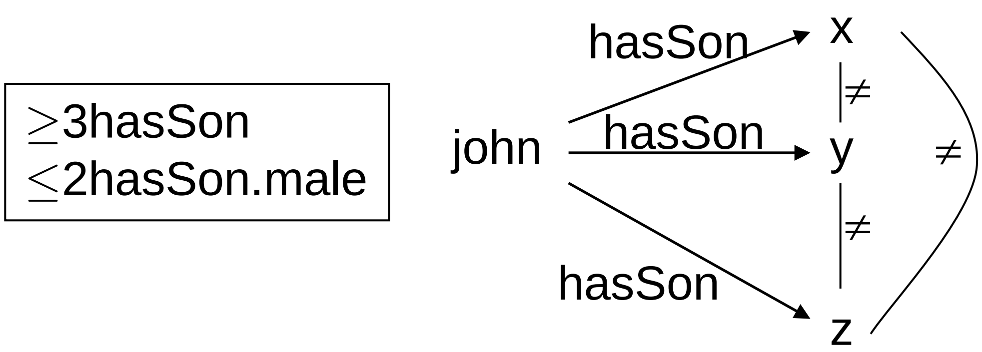
# Example (3): choose

**$\geq$3hasSon(john)**

**$\leq$2hasSon.male(john)**

**Is this contradictory?**

**No, because the following tableau is complete.**



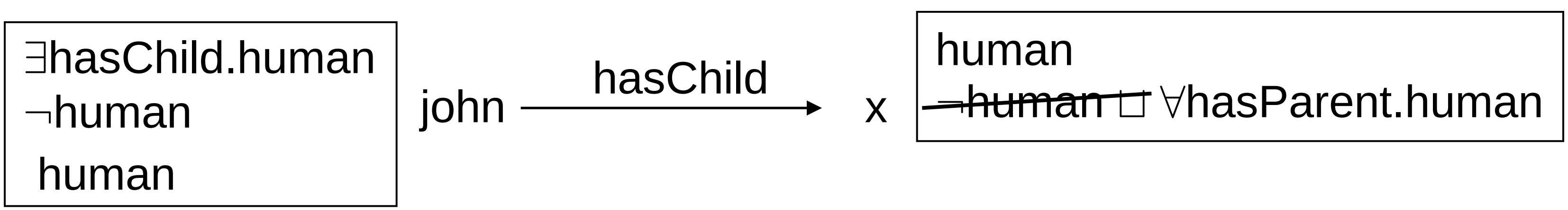
# Example (4): inverse roles

**∃hasChild.human(john)**

**human ⊑ ∀hasParent.human**

**hasChild ⊑ hasParent⁻**

**zu zeigen: human(john)**




john is hP⁻-predecessor of x, hence hP-neighbor of x

# Example (5): Transitivity and Blocking

**$\text{human} \sqsubseteq \exists \text{hasFather}.\top$**

**$\text{human} \sqsubseteq \forall \text{hasAncestor}.\text{human}$**

**$\text{hasFather} \sqsubseteq \text{hasAncestor}$ Trans(hasAncestor)**

**human(john)**

**Does this entail $\leq 1 \text{hasFather}.\top(\text{john})$?**

**Negation: $\geq 2 \text{hasFather}.\top(\text{john})$**

# Example (5): Transitivity and Blocking

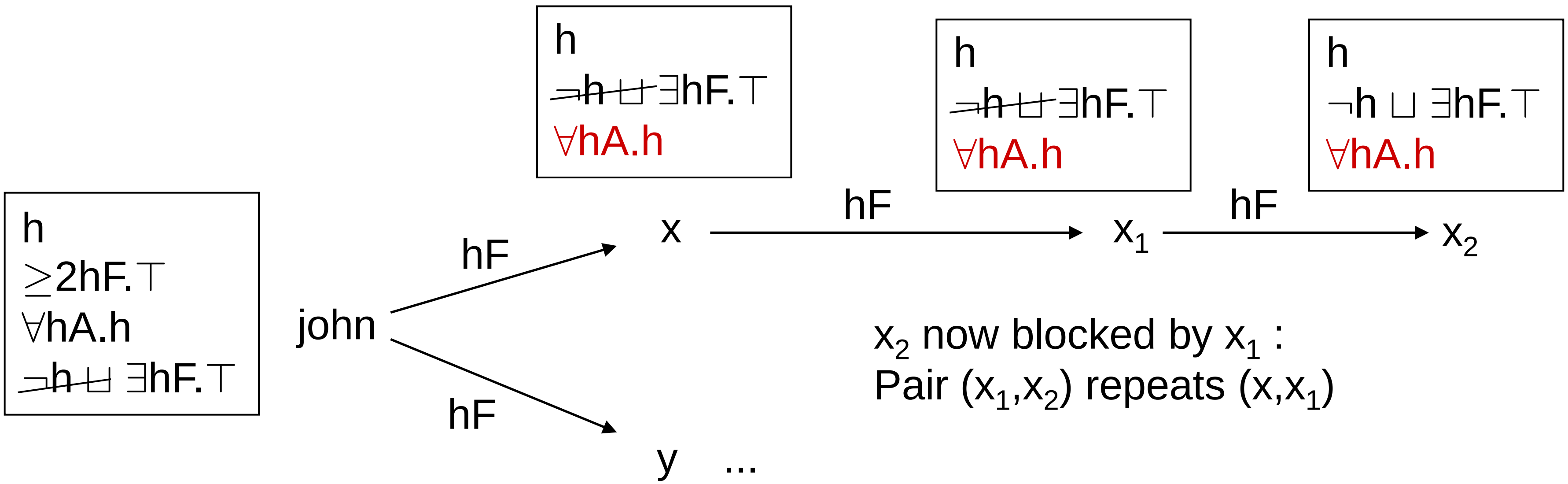
KNO.E.SIS

**human ⊑ ∃hasFather.⊤**

**hasFather ⊑ hasAncestor** **Trans(hasAncestor)**

**∀hasAncestor.human(john)**

**human(john)** **≥2hasFather.⊤(john)**

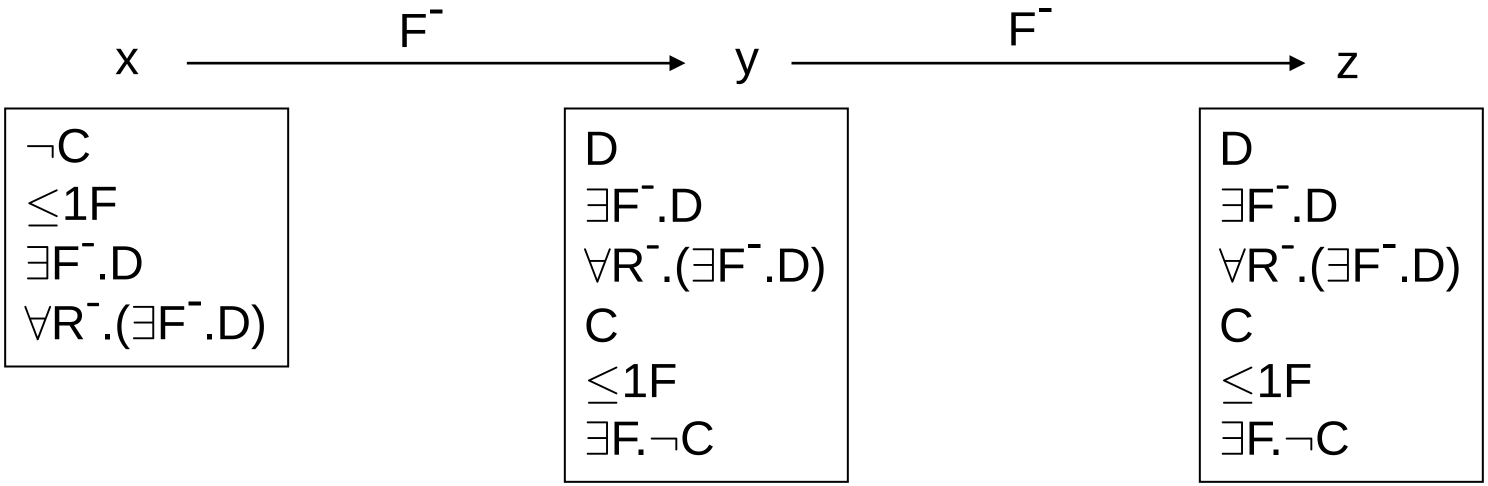



$x_2$ now blocked by $x_1$ :
Pair $(x_1,x_2)$ repeats $(x,x_1)$

same as branch above

# Example (6): Pairwise Blocking

**$\neg C \sqcap (\leq 1F) \sqcap \exists F^{-}.D \sqcap \forall R^{-}.(\exists F^{-}.D)$, where**
**$D = C \sqcap (\leq 1F) \sqcap \exists F.\neg C$, Trans(R), and $F \sqsubseteq R$,**
**is not satisfiable.**



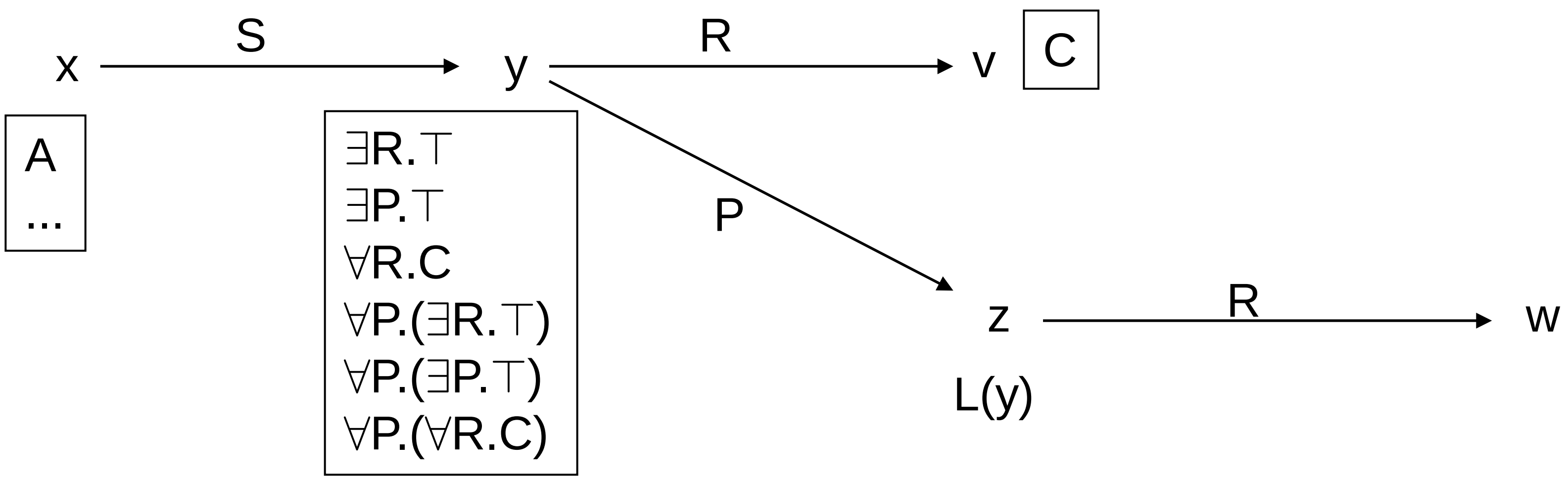

Without pairwise blocking, z would be blocked, which shouldn't happen:
Expansion of $\exists F.\neg C$ yields $\neg C$ for node y as required.

# Example (7): Dynamic Blocking

**$A \sqcap \exists S.(\exists R.\top \sqcap \exists P.\top \sqcap \forall R.C \sqcap \forall P.(\exists R.\top) \sqcap \forall P.(\forall R.C) \sqcap \forall P.(\exists P.\top))$**

**with $C = \forall R^{-}.(\forall P^{-}.(\forall S^{-}.\neg A))$ and Trans(P), is not satisfiable.**

**Part of the tableau:**




At this stage, z would be blocked by y (assuming the presence of another pair). However, when C from v is expanded, z becomes unblocked, which is necessary in order to label w with C which in turn labels x with $\neg$A, yielding the required contradiction.

# Tableaux Reasoners

- **Fact++**
  - **http://owl.man.ac.uk/factplusplus/**
- **Pellet**
  - **http://www.mindswap.org/2003/pellet/index.shtml**
- **RacerPro**
  - **http://www.sts.tu-harburg.de/~r.f.moeller/racer/**

# Contents

- OWL – Basic Ideas
- OWL As the Description Logic SROIQ(D)
- Different Perspectives on OWL
- OWL Semantics
- OWL Profiles
- Proof Theory
- Tools

# OWL tools (incomplete listing)

Reasoner:
- OWL 2 DL:
  - Pellet http://clarkparsia.com/pellet/
  - HermiT http://www.hermit-reasoner.com/
- OWL 2 EL:
  - CEL http://code.google.com/p/cel/
- OWL 2 RL:
  - essentially any rule engine
- OWL 2 QL:
  - essentially any SQL engine (with a bit of query rewriting on top)

Editors:
- Protégé
- NeOn Toolkit
- TopBraid Composer

# Main References

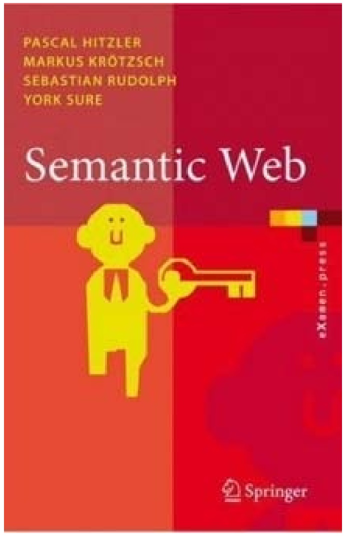

- W3C OWL Working Group, OWL 2 Web Ontology Language: Document Overview. http://www.w3.org/TR/owl2-overview/

- Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter Patel-Schneider, Sebastian Rudolph, OWL 2 Web Ontology Language: Primer. http://www.w3.org/TR/owl2-primer/

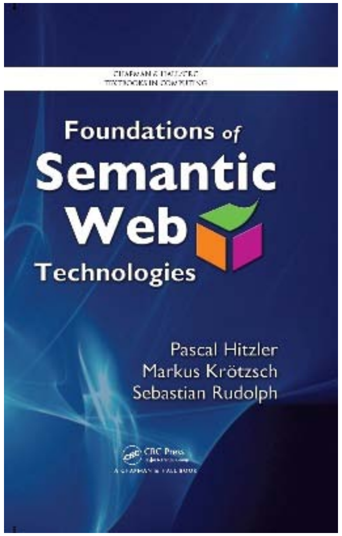
- Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, Peter F. Patel-Schneider, The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, 2nd edition, 2007.

# Main References – Textbooks

- **Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, York Sure, Semantic Web – Grundlagen. Springer, 2008. http://www.semantic-web-grundlagen.de/ (In German.) (Does not cover OWL 2.)**




- **Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009. http://www.semantic-web-book.org/wiki/FOST (Ask for a flyer from us.)**



# Further References

- DL complexity calculator: http://www.cs.man.ac.uk/~ezolin/dl/

- Markus Krötzsch, Sebastian Rudolph, Pascal Hitzler, Description Logic Rules. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, Nikos Avouris, eds.: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08), pp. 80–84. IOS Press 2008.

- Markus Krötzsch, Sebastian Rudolph, Pascal Hitzler, ELP: Tractable Rules for OWL 2. In: Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, Krishnaprasad Thirunarayan (eds.), The Semantic Web - ISWC 2008, 7th International Semantic Web Conference. Springer Lecture Notes in Computer Science Vol. 5318, 2008, pp. 649-664.

Thanks!

http://www.semantic-web-book.org/page/ISWC2010_Tutorial

# OWL 2 and Rules

–

# Optional Part, If Enough Time

# Main References Optional Part

**Main References:**


- Markus Krötzsch, Sebastian Rudolph, Pascal Hitzler, Description Logic Rules. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, Nikos Avouris, eds.: Proceedings of the 18th European Conference on Artificial Intelligence (**ECAI-08**), pp. 80–84. IOS Press 2008.
- Markus Krötzsch, Sebastian Rudolph, Pascal Hitzler, ELP: Tractable Rules for OWL 2. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, Krishnaprasad Thirunarayan, eds.: Proceedings of the 7th International Semantic Web Conference (**ISWC-08**), pp. 649–664. Springer 2008.

# Contents

- **Motivation: OWL and Rules**
- **Preliminaries: Datalog**

Intro

- **More rules than you ever need: SWRL**
- **Retaining decidability I: DL-safety**
- **Retaining decidability II: DL Rules**

Extending OWL with Rules

- **The rules hidden in OWL 2: SROIQ Rules**
- **Retaining tractability I: OWL 2 EL Rules**
- **Retaining tractability II: DLP 2**

Rules inside OWL

- **Retaining tractability III: ELP**

putting it all together

# Contents

Kno.e.sis

- **Motivation: OWL and Rules**
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# Motivation: OWL and Rules

- Rules (mainly, logic programming) as alternative ontology modelling paradigm.
- Similar tradition, and in use in practice (e.g. F-Logic)

- Ongoing: W3C RIF working group
  - Rule Interchange Format
  - based on Horn-logic
  - language standard forthcoming 2009

- Seek: Integration of rules paradigm with ontology paradigm
  - Here: Tight Integration in the tradition of OWL
  - Foundational obstacle: reasoning efficiency / decidability [naive combinations are undecidable]

# Contents

- Motivation: OWL and Rules
- **Preliminaries: Datalog**

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# Preliminaries: Datalog

- **Essentially Horn-rules without function symbols**

  **general form of the rules:**

  $p_1(x_1,\dots,x_n) \wedge \dots \wedge p_m(y_1,\dots,y_k) \rightarrow q(z_1,\dots,z_j)$

  **body → head**

  **semantics either as in predicate logic**
  **or as Herbrand semantics (see next slide)**

- **decidable**
- **polynomial data complexity (in number of facts)**
- **combined (overall) complexity: ExpTime**
- **combined complexity is P if the number of variables per rule is globally bounded**

# Datalog semantics example

- **Example:**
  **$p(x) \rightarrow q(x)$**
  **$q(x) \rightarrow r(x)$**
  **$\rightarrow p(a)$**

- **predicate logic semantics:**

  **$(\forall x)\ (p(x) \rightarrow r(x))$**
  **and**
  **$(\forall x)\ (\neg r(x) \rightarrow \neg p(x))$**
  **are logical consequences**

  **q(a) and r(a)**
  **are logical consequences**

- **Herbrand semantics**

  **those on the left are not logical consequences**

  **q(a) and r(a)**
  **are logical consequences**

  **material implication:**
  **apply only to known constants**

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- **More rules than you ever need: SWRL**
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# More rules than you ever need: SWRL

- **Union of OWL DL with (binary) function-free Horn rules (with binary Datalog rules)**

- **undecidable**
- **no native tools available**

- **rather an overarching formalism**

- **see http://www.w3.org/Submission/SWRL/**

# SWRL example (running example)

NutAllergic(sebastian)

NutProduct(peanutOil)

$\exists$orderedDish.ThaiCurry(sebastian)

ThaiCurry $\sqsubseteq$ $\exists$contains.{peanutOil}

$\top$ $\sqsubseteq$ $\forall$orderedDish.Dish

NutAllergic(x) $\land$ NutProduct(y) $\rightarrow$ dislikes(x,y)

dislikes(x,z) $\land$ Dish(y) $\land$ contains(y,z) $\rightarrow$ dislikes(x,y)

orderedDish(x,y) $\land$ dislikes(x,y) $\rightarrow$ Unhappy(x)

# SWRL example (running example)

$NutAllergic(sebastian)$

$NutProduct(peanutOil)$

$\exists orderedDish.ThaiCurry(sebastian)$

$ThaiCurry \sqsubseteq \exists contains.\{peanutOil\}$

$\top \sqsubseteq \forall orderedDish.Dish$

$NutAllergic(x) \land NutProduct(y) \rightarrow dislikes(x,y)$

$dislikes(x,z) \land Dish(y) \land contains(y,z) \rightarrow dislikes(x,y)$

$orderedDish(x,y) \land dislikes(x,y) \rightarrow Unhappy(x)$

**Conclusions:**

dislikes(sebastian,peanutOil)

# SWRL example (running example)

NutAllergic(sebastian)

NutProduct(peanutOil)

$\exists$orderedDish.ThaiCurry(sebastian)

ThaiCurry $\sqsubseteq$ $\exists$contains.{peanutOil}

$\top \sqsubseteq \forall$orderedDish.Dish

orderedDish rdfs:range Dish.

NutAllergic(x) $\wedge$ NutProduct(y) $\rightarrow$ dislikes(x,y)

dislikes(x,z) $\wedge$ Dish(y) $\wedge$ contains(y,z) $\rightarrow$ dislikes(x,y)

orderedDish(x,y) $\wedge$ dislikes(x,y) $\rightarrow$ Unhappy(x)

**Conclusions:**

**dislikes(sebastian,peanutOil)**

**orderedDish(sebastian,$y_s$)**

**ThaiCurry($y_s$)**

**Dish($y_s$)**

# SWRL example (running example)

**NutAllergic(sebastian)**
**NutProduct(peanutOil)**
**∃orderedDish.ThaiCurry(sebastian)**

**ThaiCurry ⊑ ∃contains.{peanutOil}**
**⊤ ⊑ ∀orderedDish.Dish**

**NutAllergic(x) ∧ NutProduct(y) → dislikes(x,y)**
**dislikes(x,z) ∧ Dish(y) ∧ contains(y,z) → dislikes(x,y)**
**orderedDish(x,y) ∧ dislikes(x,y) → Unhappy(x)**

**Conclusions:**
**dislikes(sebastian,peanutOil)**
**orderedDish(sebastian,$y_s$)**
**ThaiCurry($y_s$)**
**Dish($y_s$)**
**contains($y_s$,peanutOil)**

# SWRL example (running example)

**NutAllergic(sebastian)**
**NutProduct(peanutOil)**
**∃orderedDish.ThaiCurry(sebastian)**

**ThaiCurry ⊑ ∃contains.{peanutOil}**
**⊤ ⊑ ∀orderedDish.Dish**

**NutAllergic(x) ∧ NutProduct(y) → dislikes(x,y)**
**dislikes(x,z) ∧ Dish(y) ∧ contains(y,z) → dislikes(x,y)**
**orderedDish(x,y) ∧ dislikes(x,y) → Unhappy(x)**

**Conclusions:**
**dislikes(sebastian,peanutOil)**
**orderedDish(sebastian,$y_s$)**
**ThaiCurry($y_s$)**
**Dish($y_s$)**
**contains($y_s$,peanutOil)**
**dislikes(sebastian,$y_s$)**

# SWRL example (running example)

$NutAllergic(sebastian)$
$NutProduct(peanutOil)$
$\exists orderedDish.ThaiCurry(sebastian)$

$ThaiCurry \sqsubseteq \exists contains.\{peanutOil\}$
$\top \sqsubseteq \forall orderedDish.Dish$

$NutAllergic(x) \land NutProduct(y) \rightarrow dislikes(x,y)$
$dislikes(x,z) \land Dish(y) \land contains(y,z) \rightarrow dislikes(x,y)$
$orderedDish(x,y) \land dislikes(x,y) \rightarrow Unhappy(x)$

**Conclusions:**

$dislikes(sebastian,peanutOil)$
$orderedDish(sebastian,y_s)$
$ThaiCurry(y_s)$
$Dish(y_s)$
$contains(y_s,peanutOil)$
$dislikes(sebastian,y_s)$
$Unhappy(sebastian)$

# SWRL example (running example)

$\text{NutAllergic(sebastian)}$

$\text{NutProduct(peanutOil)}$

$\exists\text{orderedDish.ThaiCurry(sebastian)}$

$\text{ThaiCurry} \sqsubseteq \exists\text{contains.\{peanutOil\}}$

$\top \sqsubseteq \forall\text{orderedDish.Dish}$

$\text{NutAllergic(x)} \wedge \text{NutProduct(y)} \rightarrow \text{dislikes(x,y)}$

$\text{dislikes(x,z)} \wedge \text{Dish(y)} \wedge \text{contains(y,z)} \rightarrow \text{dislikes(x,y)}$

$\text{orderedDish(x,y)} \wedge \text{dislikes(x,y)} \rightarrow \text{Unhappy(x)}$

**Conclusion: Unhappy(sebastian)**

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- **Retaining decidability I: DL-safety**
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# Retaining decidability I: DL-safety

- **Reinterpret SWRL rules:**
  **Rules apply only to individuals which are explicitly given in the knowledge base.**
  - **Herbrand-style way of interpreting them**

- **OWL DL + DL-safe SWRL is decidable**
- **Native support e.g. by KAON2 and Pellet**

- See e.g. Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. Journal of Web Semantics 3(1):41–60, 2005.

# DL-safe SWRL example

**NutAllergic(sebastian)**
**NutProduct(peanutOil)**
**$\exists$orderedDish.ThaiCurry(sebastian)**

**ThaiCurry $\sqsubseteq$ $\exists$contains.{peanutOil}**
**$\top$ $\sqsubseteq$ $\forall$orderedDish.Dish**

DL-safe {
**NutAllergic(x) $\wedge$ NutProduct(y) $\rightarrow$ dislikes(x,y)**
**dislikes(x,z) $\wedge$ Dish(y) $\wedge$ contains(y,z) $\rightarrow$ dislikes(x,y)**
**orderedDish(x,y) $\wedge$ dislikes(x,y) $\rightarrow$ Unhappy(x)**

**Unhappy(sebastian) can*not* be concluded**

# DL-safe SWRL example

**NutAllergic(sebastian)**

**NutProduct(peanutOil)**

**∃orderedDish.ThaiCurry(sebastian)**

**ThaiCurry ⊑ ∃contains.{peanutOil}**

**⊤ ⊑ ∀orderedDish.Dish**

DL-safe {

**NutAllergic(x) ∧ NutProduct(y) → dislikes(x,y)**

**dislikes(x,z) ∧ Dish(y) ∧ contains(y,z) → dislikes(x,y)**

**orderedDish(x,y) ∧ dislikes(x,y) → Unhappy(x)**

**Conclusions:**

**dislikes(sebastian,peanutOil)**

**orderedDish(sebastian,$y_s$)**

**ThaiCurry($y_s$)**

**Dish($y_s$)**

**contains($y_s$,peanutOil)**

~~dislikes(sebastian,$y_s$)~~

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- **Retaining decidability II: DL Rules**

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# Retaining decidability II: DL Rules

- **General idea:**
  **Find out which rules can be encoded in OWL (2 DL) anyway**

- **$\text{Man}(x) \land \text{hasBrother}(x,y) \land \text{hasChild}(y,z) \rightarrow \text{Uncle}(x)$**
  - **$\text{Man} \sqcap \exists \text{hasBrother}.\exists \text{hasChild}.\top \sqsubseteq \text{Uncle}$**

- **$\text{ThaiCurry}(x) \rightarrow \exists \text{contains}.\text{FishProduct}(x)$**
  - **$\text{ThaiCurry} \sqsubseteq \exists \text{contains}.\text{FishProduct}$**

- **$\text{kills}(x,x) \rightarrow \text{suicide}(x)$** **$\text{suicide}(x) \rightarrow \text{kills}(x,x)$**
  - **$\exists \text{kills}.\text{Self} \sqsubseteq \text{suicide}$** **$\text{suicide} \sqsubseteq \exists \text{kills}.\text{Self}$**

**Note: with these two axioms,**
***suicide* is basically the same as *kills***

# DL Rules: more examples

- $NutAllergic(x) \wedge NutProduct(y) \rightarrow dislikes(x,y)$
  - NutAllergic ≡ ∃nutAllergic.Self
    NutProduct ≡ ∃nutProduct.Self
    nutAllergic o U o nutProduct ⊑ dislikes

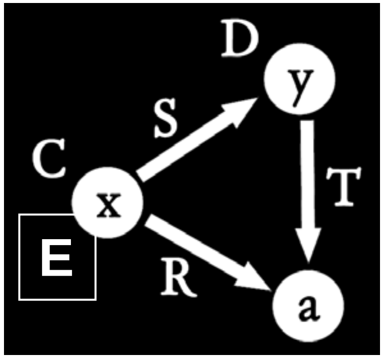
- $dislikes(x,z) \wedge Dish(y) \wedge contains(y,z) \rightarrow dislikes(x,y)$
  - Dish ≡ ∃dish.Self
    dislikes o contains$^{-}$ o dish ⊑ dislikes

- $worksAt(x,y) \wedge University(y) \wedge supervises(x,z) \wedge PhDStudent(z) \rightarrow professorOf(x,z)$
  - ∃worksAt.University ≡ ∃worksAtUniversity.Self
    PhDStudent ≡ ∃phDStudent.Self
    worksAtUniversity o supervises o phDStudent ⊑ professorOf

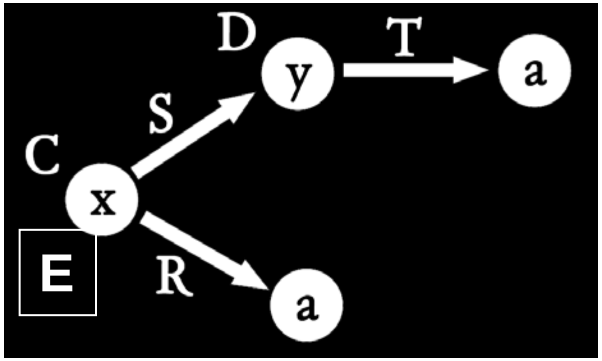
# DL Rules: definition

- **Tree-shaped bodies**
- **First argument of the conclusion is the root**

- $C(x) \wedge R(x,a) \wedge S(x,y) \wedge D(y) \wedge T(y,a) \rightarrow E(x)$
  - $C \sqcap \exists R.\{a\} \sqcap \exists S.(D \sqcap \exists T.\{a\}) \sqsubseteq E$




**duplicating nominals is ok**

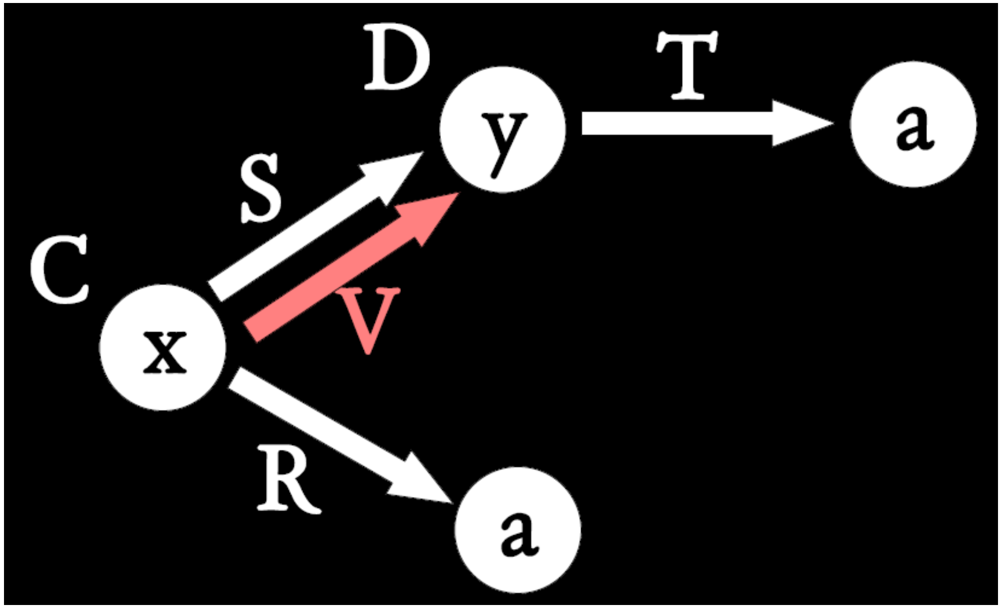


# DL Rules: definition

- **Tree-shaped bodies**
- **First argument of the conclusion is the root**

- **C(x) ∧ R(x,a) ∧ S(x,y) ∧ D(y) ∧ T(y,a) → V(x,y)**

**C ⊓ ∃R.{a} ⊑ ∃R1.Self**
**D ⊓ ∃T.{a} ⊑ ∃R2.Self**
**R1 o S o R2 ⊑ V**



# DL Rules: definition

- Tree-shaped bodies
- First argument of the conclusion is the root

- complex classes are allowed in the rules
  - Mouse(x) $\land$ ∃hasNose.TrunkLike(y) $\rightarrow$ smallerThan(x,y)
  - ThaiCurry(x) $\rightarrow$ ∃contains.FishProduct(x)

    Note: This allows to reason with unknowns (unlike Datalog)
  - allowed class constructors depend on the chosen underlying description logic!

# DL Rules: definition

**Given a description logic $\mathcal{D}$,**

**the language $\mathcal{D}$ Rules consists of**

- **all axioms expressible in $\mathcal{D}$,**
- **plus all rules with**
  - **tree-shaped bodies, where**
  - **the first argument of the conclusion is the root, and**
  - **complex classes from $\mathcal{D}$ are allowed in the rules.**
  - **<plus possibly some restrictions concerning e.g. the use of simple roles – depending on $\mathcal{D}$>**

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- **The rules hidden in OWL 2: SROIQ Rules**
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# The rules hidden in OWL 2: SROIQ Rules

- **N2ExpTime complete**

- **In fact, SROIQ Rules can be translated into SROIQ**
  **i.e. they don't add expressivity.**

  **Translation is polynomial.**

- **SROIQ Rules are essentially helpful syntactic sugar for OWL 2.**

# SROIQ Rules example

**NutAllergic(sebastian)**

**NutProduct(peanutOil)**

**$\exists$orderedDish.ThaiCurry(sebastian)**

**ThaiCurry $\sqsubseteq$ $\exists$contains.{peanutOil}**

**$\top$ $\sqsubseteq$ $\forall$orderedDish.Dish**

**NutAllergic(x) $\wedge$ NutProduct(y) $\rightarrow$ dislikes(x,y)**

**dislikes(x,z) $\wedge$ Dish(y) $\wedge$ contains(y,z) $\rightarrow$ dislikes(x,y)**

**orderedDish(x,y) $\wedge$ dislikes(x,y) $\rightarrow$ Unhappy(x)**

**!not a SROIQ Rule!**

# SROIQ Rules normal form

- Each SROIQ Rule can be written ("linearised") such that
  - the body-tree is linear,
  - if the head is of the form R(x,y), then y is the leaf of the tree, and
  - if the head is of the form C(x), then the tree is only the root.

- worksAt(x,y) ∧ University(y) ∧ supervises(x,z) ∧ PhDStudent(z) → professorOf(x,z)
  - ∃worksAt.University(x) ∧ supervises(x,z) ∧ PhDStudent(z) → professorOf(x,z)

- C(x) ∧ R(x,a) ∧ S(x,y) ∧ D(y) ∧ T(y,a) → V(x,y)
  - (C ⊓ ∃R.{a})(x) ∧ S(x,y) ∧ (D ⊓ ∃T.{a})(y) → V(x,y)

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- **Retaining tractability I: OWL 2 EL Rules**
- Retaining tractability II: DLP 2

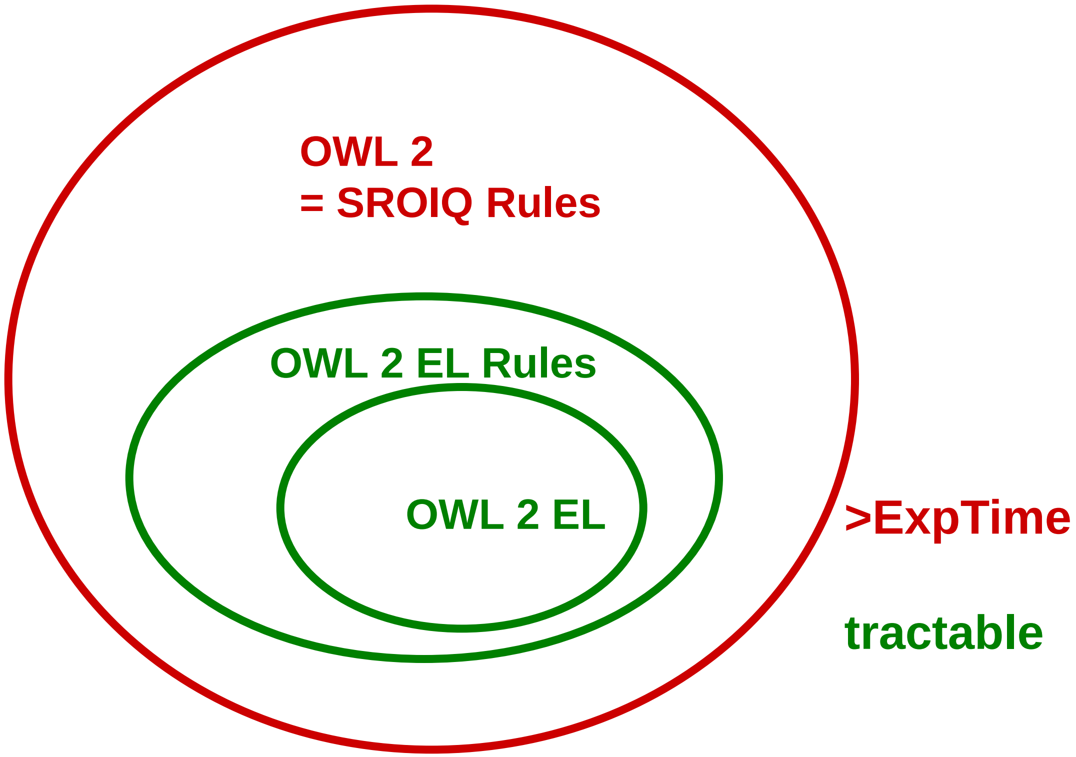
Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# Retaining tractability I: OWL 2 EL Rules

- **EL++ Rules are PTime complete**
- **EL++ Rules offer expressivity which is not readily available in EL++.**



# OWL 2 EL Rules: normal form

- Every EL++ Rule can be converted into a normal form,where
  - occurring classes in the rule body are either atomic or nominals,
  - all variables in a rule's head occur also in its body, and
  - rule heads can only be of one of the forms A(x), ∃R.A(x), R(x,y), where A is an atomic class or a nominal or ⊤ or ⊥.

- Translation is polynomial.

- ∃worksAt.University(x) ∧ supervises(x,z) ∧ PhDStudent(z) → professorOf(x,z)
  - worksAt(x,y) ∧ University(y) ∧ supervises(x,z) ∧ PhDStudent(z) → professorOf(x,z)

- ThaiCurry(x) → ∃contains.FishProduct(x)

# OWL 2 EL Rules in a nutshell

Essentially, OWL 2 EL Rules is

- Binary Datalog with tree-shaped rule bodies,
- extended by
  - occurrence of nominals as atoms and
  - existential class expressions in the head.

- The existentials really make the difference.

- Arguably the better alternative to OWL 2 EL (aka EL++)?
  - (which is covered anyway)

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- **Retaining tractability II: DLP 2**

Rules inside OWL

- Retaining tractability III: ELP

putting it all together

# Retaining tractability II: DLP 2

- DLP 2 is
  - DLP (aka OWL 2 RL) extended with
  - DL rules, which use
    - left-hand-side class expressions in the bodies and
    - right-hand-side class expressions in the head.

- Polynomial transformation into 5-variable Horn rules.

- PTime.

- Quite a bit more expressive than DLP / OWL 2 RL ...

# Contents

- Motivation: OWL and Rules
- Preliminaries: Datalog

Intro

- More rules than you ever need: SWRL
- Retaining decidability I: DL-safety
- Retaining decidability II: DL Rules

Extending OWL with Rules

- The rules hidden in OWL 2: SROIQ Rules
- Retaining tractability I: OWL 2 EL Rules
- Retaining tractability II: DLP 2

Rules inside OWL

- **Retaining tractability III: ELP**

putting it all together

# Retaining tractability III: ELP

**Putting it all together:**

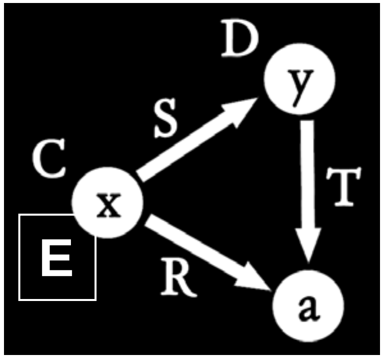
- **ELP is**
  - **OWL 2 EL Rules +**
  - **a generalisation of DL-safety +**
  - **variable-restricted DL-safe Datalog +**
  - **role conjunctions (for simple roles).**

- **PTime complete.**
- **Contains OWL 2 EL and OWL 2 RL.**
- **Covers variable-restricted Datalog.**

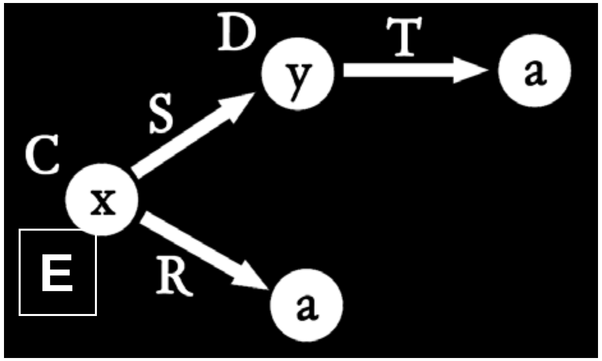
# DL-safe variables

- A generalisation of DL-safety.
- DL-safe variables are special variables which bind only to named individuals (like in DL-safe rules).
- DL-safe variables can replace individuals in EL++ rules.

- $C(x) \wedge R(x,x_s) \wedge S(x,y) \wedge D(y) \wedge T(y,x_s) \rightarrow E(x)$
  with $x_s$ a safe variable is allowed, because
  $C(x) \wedge R(x,a) \wedge S(x,y) \wedge D(y) \wedge T(y,a) \rightarrow E(x)$
  is an EL++ rule.




duplicating nominals is ok



# Variable-restricted DL-safe Datalog

- n-Datalog is Datalog, where the number of variables occurring in rules is globally bounded by n.

- complexity of n-Datalog is PTime (for fixed n)
  - (but exponential in n)

- in a sense, this is cheating.
- in another sense, this means that using a few DL-safe Datalog rules together with an EL++ rules knowledge base shouldn't really be a problem in terms of reasoning performance.

# Role conjunctions

- **orderedDish(x,y) $\wedge$ dislikes(x,y) $\rightarrow$ Unhappy(x)**

- **In fact, role conjunctions can also be added to OWL 2 DL without increase in complexity.**

- Sebastian Rudolph, Markus Krötzsch, Pascal Hitzler, Cheap Boolean Role Constructors for Description Logics. In: Steffen Hölldobler and Carsten Lutz and Heinrich Wansing (eds.), Proceedings of 11th European Conference on Logics in Artificial Intelligence (JELIA), volume 5293 of LNAI, pp. 362-374. Springer, September 2008.

# Retaining tractability III: ELP

- $\text{ELP}_n$ is
  - OWL 2 EL Rules generalised by DL-safe variables +
  - DL-safe Datalog rules with at most n variables +
  - role conjunctions (for simple roles).
- PTime complete (for fixed n).
  - exponential in n
- Contains OWL 2 EL and OWL 2 RL.
- Covers all Datalog rules with at most n variables. (!)

# ELP example

**NutAllergic(sebastian)**

**NutProduct(peanutOil)**

**$\exists$orderedDish.ThaiCurry(sebastian)**

**ThaiCurry $\sqsubseteq$ $\exists$contains.{peanutOil}**

**$\top$ $\sqsubseteq$ $\forall$orderedDish.Dish**

[okay] **NutAllergic(x) $\wedge$ NutProduct(y) $\rightarrow$ dislikes(x,y)**

**dislikes(x,z) $\wedge$ Dish(y) $\wedge$ contains(y,z) $\rightarrow$ dislikes(x,y)**

**orderedDish(x,y) $\wedge$ dislikes(x,y) $\rightarrow$ Unhappy(x)**

[okay – role conjunction]

**not an EL++ rule**

# ELP example

- **dislikes(x,z) ∧ Dish(y) ∧ contains(y,z) → dislikes(x,y) as SROIQ rule translates to**

  **Dish ≡ ∃dish.Self**
  **dislikes o contains⁻ o dish ⊑ dislikes**

  **but we don't have inverse roles in ELP!**

- **solution: make z a DL-safe variable:**

  **dislikes(x,!z) ∧ Dish(y) ∧ contains(y,!z) → dislikes(x,y)**

  **this is fine ☺**

# DL-safe SWRL example

NutAllergic(sebastian)
NutProduct(peanutOil)
∃orderedDish.ThaiCurry(sebastian)

ThaiCurry ⊑ ∃contains.{peanutOil}
⊤ ⊑ ∀orderedDish.Dish

NutAllergic(x) ∧ NutProduct(y) → dislikes(x,y)
dislikes(x,!z) ∧ Dish(y) ∧ contains(y,!z) → dislikes(x,y)
orderedDish(x,y) ∧ dislikes(x,y) → Unhappy(x)

**Conclusions:**
dislikes(sebastian,peanutOil)
orderedDish(sebastian,$y_s$)
ThaiCurry($y_s$)
Dish($y_s$)
contains($y_s$,peanutOil)
dislikes(sebastian,$y_s$)

# ELP example

**NutAllergic(sebastian)**
**NutProduct(peanutOil)**
**$\exists$orderedDish.ThaiCurry(sebastian)**

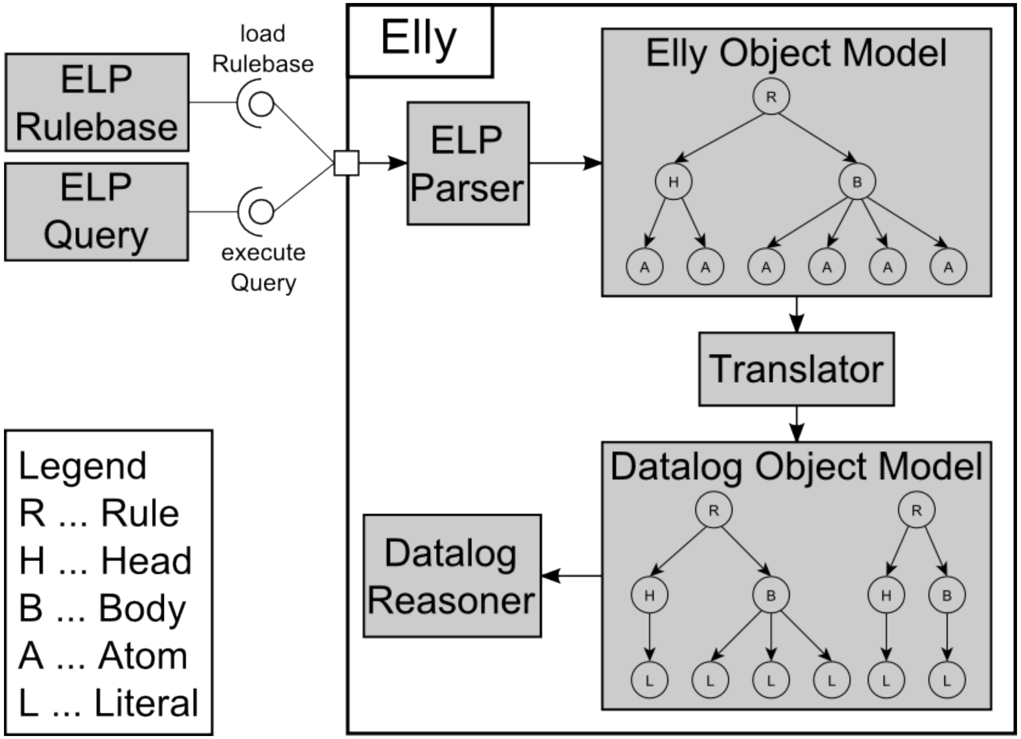
**ThaiCurry $\sqsubseteq$ $\exists$contains.{peanutOil}**
**$\top \sqsubseteq \forall$orderedDish.Dish**

**NutAllergic(x) $\wedge$ NutProduct(y) $\rightarrow$ dislikes(x,y)**
**dislikes(x,!z) $\wedge$ Dish(y) $\wedge$ contains(y,!z) $\rightarrow$ dislikes(x,y)**
**orderedDish(x,y) $\wedge$ dislikes(x,y) $\rightarrow$ Unhappy(x)**

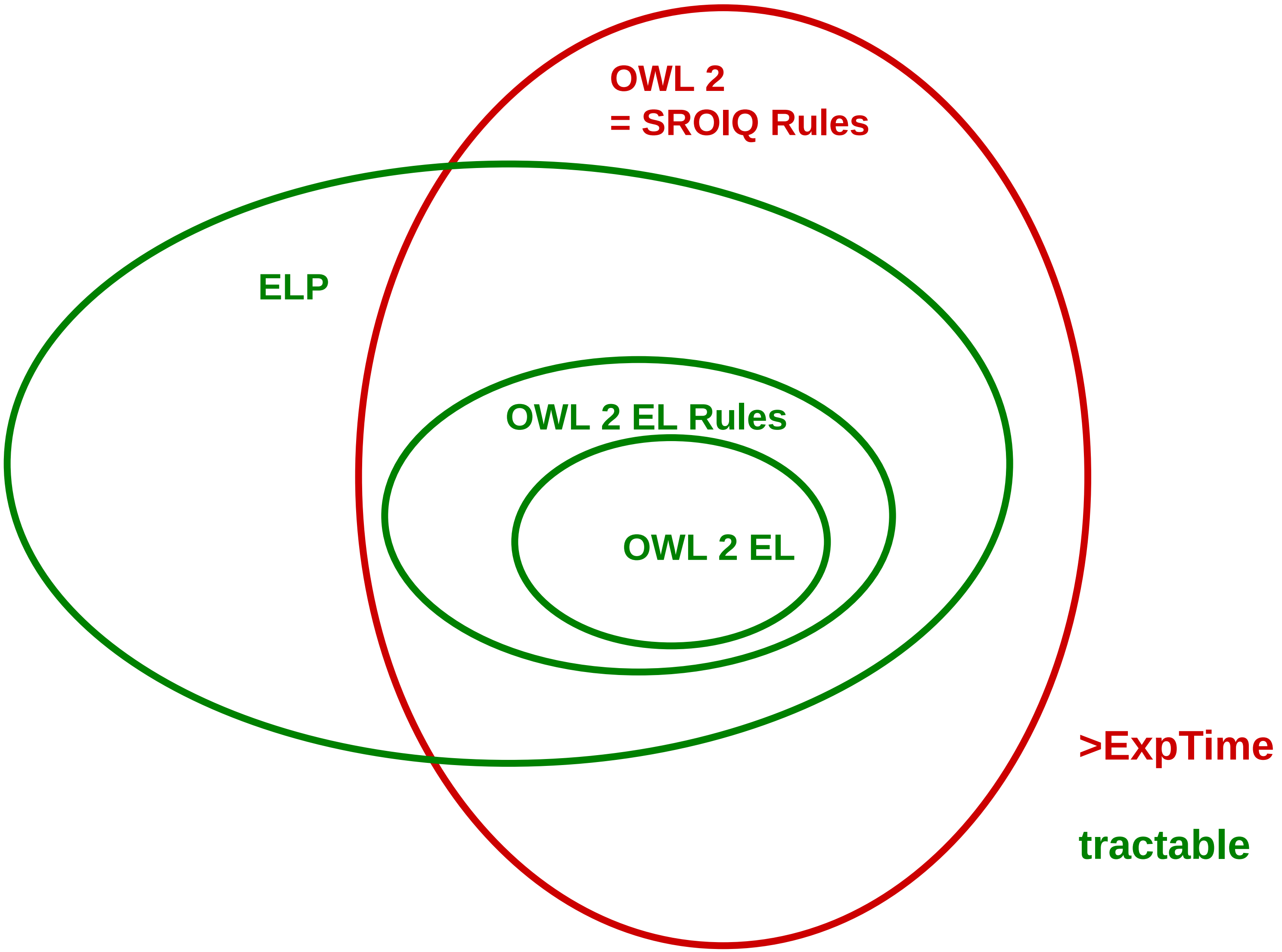
**Conclusion: Unhappy(sebastian)**

# ELP Reasoner ELLY

- Implementation currently being finalised.
- Based on IRIS Datalog reasoner.
- In cooperation with STI Innsbruck (Barry Bishop, Daniel Winkler, Gulay Unel).



# The Big Picture

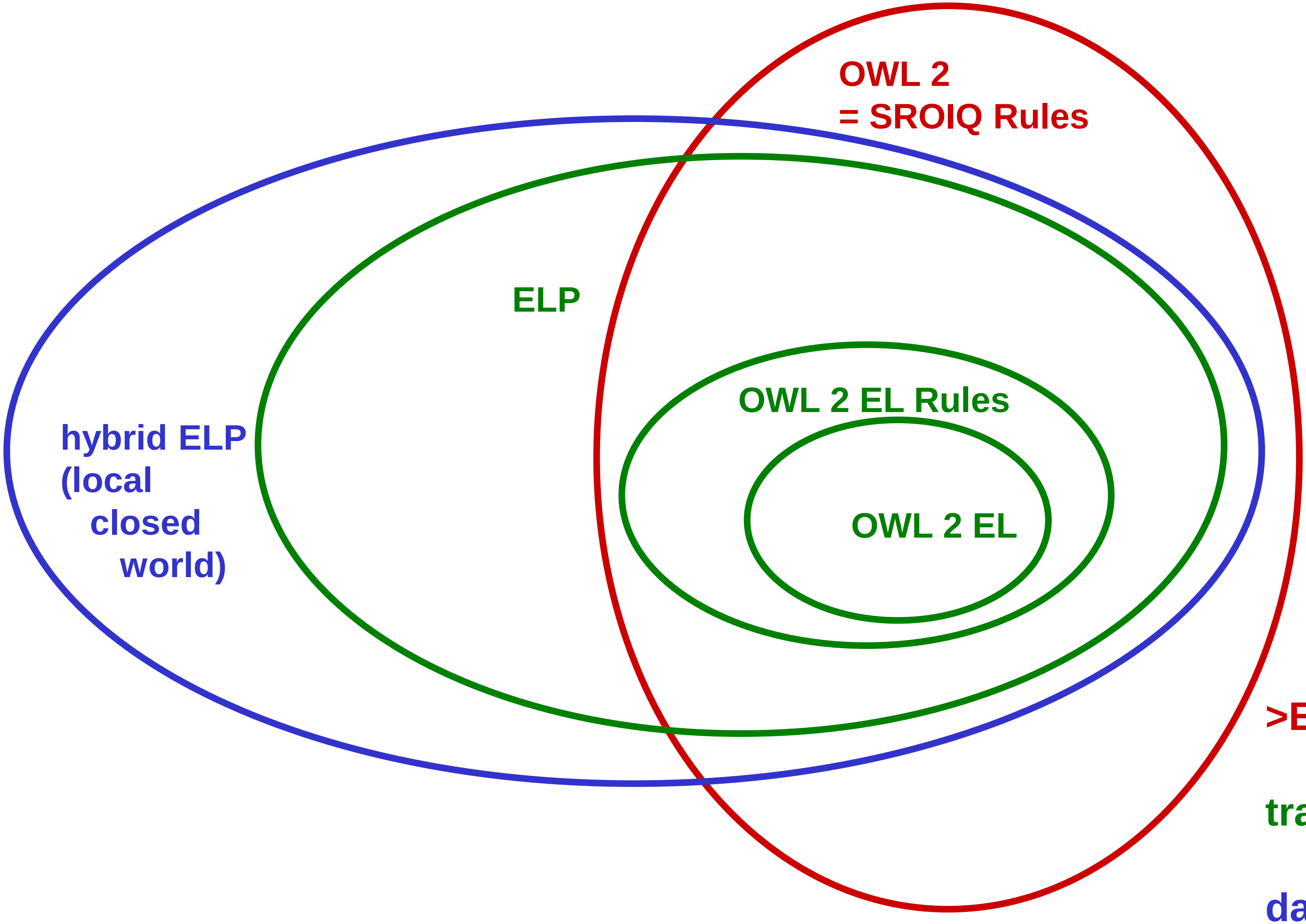


# Closed World and ELP

- **There's an extension of ELP using (non-monotonic) closed-world reasoning – based on a well-founded semantics for hybrid MKNF knowledge bases.**

- Matthias Knorr, Jose Julio Alferes, Pascal Hitzler, A Coherent Well-founded model for Hybrid MKNF knowledge bases. In: Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, Nikos Avouris (eds.), Proceedings of the 18th European Conference on Artificial Intelligence, ECAI2008, Patras, Greece, July 2008. IOS Press, 2008, pp. 99-103.

# The Big Picture II



Thanks!

http://www.semantic-web-book.org/page/ISWC2010_Tutorial

# References OWL and Rules

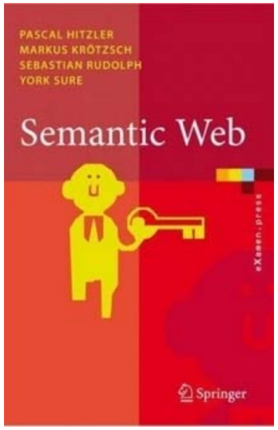

- Markus Krötzsch, Sebastian Rudolph, Pascal Hitzler, Description Logic Rules. In Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, Nikos Avouris, eds.: Proceedings of the 18th European Conference on Artificial Intelligence (ECAI-08), pp. 80–84. IOS Press 2008.
- Markus Krötzsch, Sebastian Rudolph, Pascal Hitzler, ELP: Tractable Rules for OWL 2. In Amit Sheth, Steffen Staab, Mike Dean, Massimo Paolucci, Diana Maynard, Timothy Finin, Krishnaprasad Thirunarayan, eds.: Proceedings of the 7th International Semantic Web Conference (ISWC-08), pp. 649–664. Springer 2008.
- http://www.w3.org/Submission/SWRL/
- Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with Rules. Journal of Web Semantics 3(1):41–60, 2005.

# References OWL and Rules


- Sebastian Rudolph, Markus Krötzsch, Pascal Hitzler, Cheap Boolean Role Constructors for Description Logics. In: Steffen Hölldobler and Carsten Lutz and Heinrich Wansing (eds.), Proceedings of 11th European Conference on Logics in Artificial Intelligence (JELIA), volume 5293 of LNAI, pp. 362-374. Springer, September 2008.
- Matthias Knorr, Jose Julio Alferes, Pascal Hitzler, A Coherent Well-founded model for Hybrid MKNF knowledge bases. In: Malik Ghallab, Constantine D. Spyropoulos, Nikos Fakotakis, Nikos Avouris (eds.), Proceedings of the 18th European Conference on Artificial Intelligence, ECAI2008, Patras, Greece, July 2008. IOS Press, 2008, pp. 99-103.

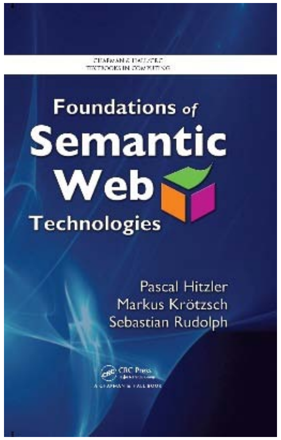
# See also our books

- **Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph, York Sure,**
  **Semantic Web – Grundlagen. Springer, 2008.**
  **http://www.semantic-web-grundlagen.de/**




- **Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph,**
  **Foundations of Semantic Web Technologies.**
  **Chapman & Hall/CRC, 2009.**
  **http://www.semantic-web-book.org/wiki/FOST**

  **(Grab a flyer.)**



# A Practical Introduction to Ontologies & OWL

Tutorial ISWC 2010

Bernardo Cuenca Grau, Birte Glimm, Pascal Hitzler, Héctor Pérez-Urbina

Material adapted from the Protégé OWL Tutorial originally developed by the BHIG group at the University of Manchester

The University of Manchester

Creative Commons License

# Overview

- Pizzas – Card Sorting
- Protégé Introduction
- Creating a Class Hierarchy
- Consistency
- Disjointness
- Properties
- Restrictions
- Defined Classes
- Union Classes
- The Open World Assumption
- Closure

The University of Manchester
Creative Commons License

# Our Domain

- Pizzas have been used in Manchester tutorials for years.
- Pizzas were selected as a domain for several reasons:
  - They are fun
  - They are internationally known
  - They are highly compositional
  - They have a natural limit to their scope
  - They are fairly neutral
    - Although arguments still break out over representation
    - Even pizzas can do this - its an inevitable part of knowledge modelling
    - ARGUING IS NOT BAD!


Creative Commons License

# You are the Expert

- Most often it is **not** the domain expert that formalises their knowledge – because of the complexity of the modelling it is normally a specialist "knowledge engineer"
  Hopefully, as tools get easier to use, this will change
- Having access to experts is critical for most domains
- Luckily, we are all experts in Pizzas, so we just need some material to verify our knowledge…


Creative Commons License

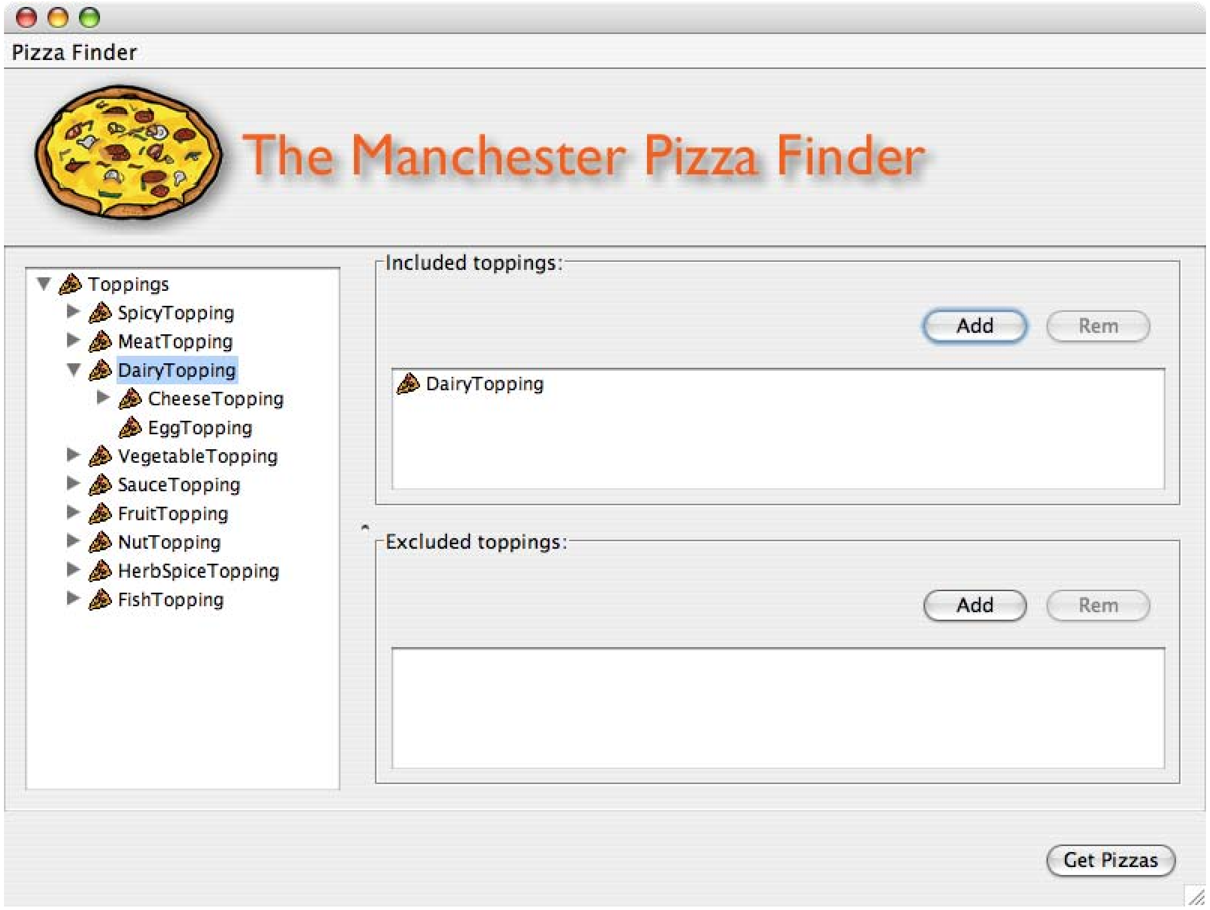
# Our Ontology

- When building an ontology we need an application in mind – ontologies should not be built for the sake of it
- Keep the application in mind when creating concepts – this should help you scope the project

- The PizzaFinder application has been developed so that you can plug your ontology in at the end of the day and see it in action


Creative Commons License

# Our Application




www.co-ode.org/downloads/pizzafinder/


Creative Commons License

# Exercise 1: Card Sorting

- You have been given a selection of pizza toppings from a takeway menu
- Group the toppings into several piles
  - What similarities and differences are there between the different piles?
  - Are there any concepts missing?
- Feel free to add you own toppings to the cards


Creative Commons License

# Card Sorting - Issues

- different viewpoints
  - Tomato – Vegetable or Fruit?
  - culinary vs biological
- Ambiguity
  - words not concepts
- Missing Knowledge
  - What is peperonata?
- multiple classifications (2+ parents)
- lots of missing categories (superclasses?)
- competency questions
  - What are we likely to want to “ask” our ontology?
  - bear the application in mind


Creative Commons License

# Editing OWL

- Editing the RDF/XML by hand is probably not recommended (as we have seen)
- Ontologies range in size, but because of their explicit nature they require verbose definitions
- Thankfully we have tools to help us reduce the syntactic complexity
- However, the tools are still in the process of trying to reduce the semantic complexity
- Building ontologies in OWL is still hard


Creative Commons License

The University of Manchester

MANCHESTER 1824

# OWL Editors

NeOn

protégé

Protégé OWL

Ontology Editor for the Semantic Web

SWOOP

MINDSWAP

ALTOVA® semanticworks™ 2006

emacs

http://www.xml.com/pub/a/2004/07/14/onto.html


Creative Commons License

# protégé...

- Is a knowledge modelling environment
- Is free, open source software
- Is developed by Stanford / Manchester
- Has a large user community (approx 30k)
- Protégé 4/4.1 Built solely on OWL modelling language
- Supports development of plugins to allow backend / interface extensions


Creative Commons License

# Exercise 2: Create Class Hierarchy

- It is helpful to be consistent in naming your entities– especially when trying to find things in your ontology
- Create a class hierarchy in an empty ontology.
- Arrange Pizza, PizzaBase, and PizzaTopping as a subclasses of Food, sort your toppings into classes under PizzaTopping
- We demo the initial steps in Protégé.
- Make sure you save your ontologies on a regular basis!

# Labels – so what?

- Humans might be able to interpret what the labels mean and how they are defined, but the computer cannot.

| | |
|---|---|
| A | Food |
| B | Pizza |
| C | PizzaBase |
| D | PizzaTopping |


Creative Commons License

# Consistency Checking

- Let's make a MeatyVegetableTopping as subclass of MeatTopping and VegetableTopping!
- We demo this
- We've just created a class that doesn't really make sense
  - What is a MeatyVegetableTopping?

- We'd like to be able to check the logical consistency of our model
- This is one of the tasks that can be done automatically by software known as a **Reasoner**
- Being able to use a reasoner is one of the main advantages of using a logic-based formalism such as OWL (and why we are using OWL-DL)


Creative Commons License

# Reasoners

- Reasoners are used to infer information that is not explicitly contained within the ontology
- You may also hear them being referred to as Classifiers
- Standard reasoner services are:
  - Consistency Checking
  - Subsumption Checking
  - Equivalence Checking
  - Instantiation Checking
- Reasoners can be used at runtime in applications as a querying mechanism (esp useful for smaller ontologies)
- We will use one during development as an ontology "compiler". A well designed ontology can be compiled to check its meaning is that intended


Creative Commons License

# Why is MeatyVegetableTopping not Inconsistent?

- We have asserted that a **MeatyVegetableTopping** is a subclass of two classes, but these classes are not disjoint
- The disjoint means nothing can be a **MeatTopping** and a **VegetableTopping** at the same time
- Try and make all direct subclasses of Thing disjoint and use the reasoner again
- We demo this


Creative Commons License

# Why is MeatyVegetableTopping Inconsistent?

- The disjoint means nothing can be a **MeatTopping** and a **VegetableTopping** at the same time
- This means that **MeatyVegetableTopping** can never contain any individuals
- The class is therefore unsatisviable- this is what we expect!
- It can be useful to create classes we expect to be inconsistent to "test" your model – often we refer to these classes as "probes" – generally it is a good idea to document them as such to avoid later confusion


Creative Commons License

# Relationships in OWL

- In OWL-DL, relationships can only be formed between Individuals or between an Individual and a data value.
  (In OWL-Full, Classes can be related, but this cannot be reasoned with)

- Relationships are formed along Properties

- We can restrict how these Properties are used:
  - Globally – by stating things about the Property itself
  - Or locally – by restricting their use for a given Class


Creative Commons License

# Creating Properties

- We often create properties using 2 standard naming patterns:
  - has… (e.g., hasColour)
  - is…Of (e.g., isTeacherOf) or other suffixes (e.g., …In …To)
- This has several advantages:
  - It is easier to find properties
  - It is easier for tools to generate a more readable form (see tooltips on the classes in the hierarchy later)
  - Inverses properties typically follow this pattern e.g., hasPart, isPartOf


Creative Commons License

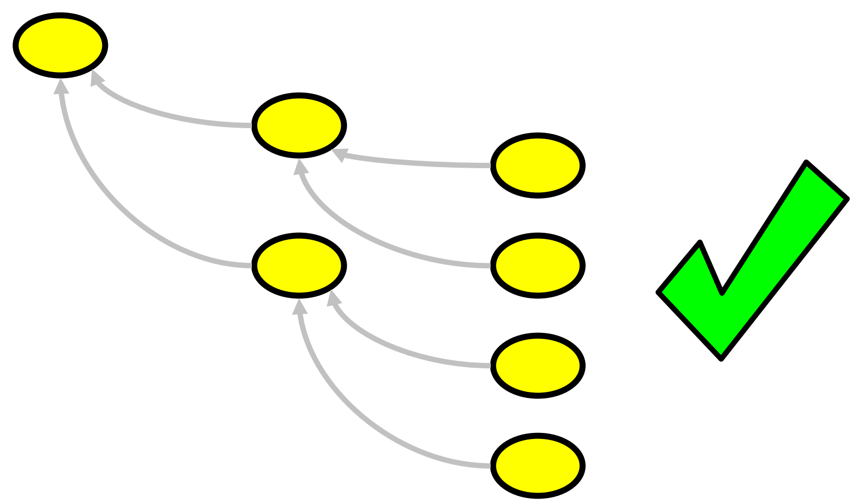
# Exercise 3: Properties

- Create a set of (object) properties that can be used to define some pizzas
- Create at least hasTopping and hasBase as subproperties of hasIngredient
- We demo the creation of properties


Creative Commons License

# Primitive Classes

- All classes in our ontology so far are Primitive
- We describe primitive pizzas
- Primitive Class = only Necessary Conditions
- They are marked as plain orange circles in the class hierarchy




Creative Commons License

# Polyhierarchies

- By the end of this tutorial we intent to create a **VegetarianPizza**
- Some of our existing Pizzas should be types of **VegetarianPizza**
- However, they could also be types of **CheeseyPizza**

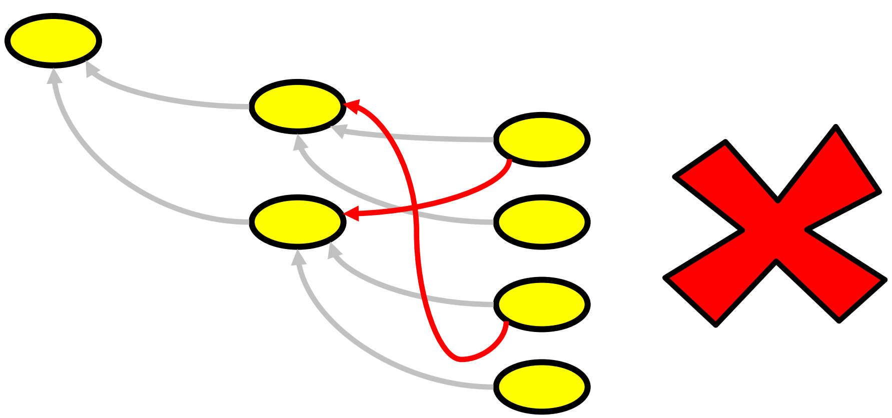
- We need to be able to give them multiple parents in a principled way
- We could just assert multiple parents like we did with **MeatyVegetableTopping** (without disjoints)

BUT…

# Asserted Polyhierarchies

We believe asserting polyhierarchies is bad

- We lose some encapsulation of knowledge
  - Why is this class a subclass of that one?
- Difficult to maintain
  - Adding new classes becomes difficult because all subclasses may need to be updated
  - Extracting from a graph is harder than from a tree



**let the reasoner do it!**


Creative Commons License

# Describing Classes using Properties

- To do this, we go back to the **Pizza** class and add some further information
- This comes in the form of Restrictions
- Restrictions are a type of anonymous class
- They describe the relationships that must hold for members (Individuals) of this class
- We create Restrictions using the Class Description Frame

- Conditions can be any kind of Class – you have already added Named superclasses in the Class Description Frame. Restrictions are a type of Anonymous Class


Creative Commons License

# Anonymous Classes

- Made up of logical expressions
  - Unions and Intersections (Or, And)
  - Complements (Not)
  - Enumerations (specified membership)
  - Restrictions (related to Property use)

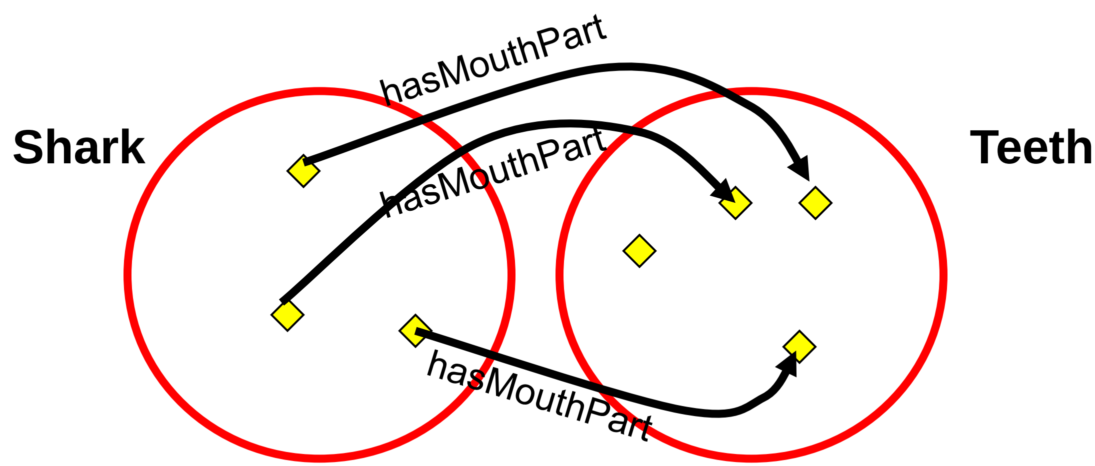
- The members of an anonymous class are the set of Individuals that satisfy its logical definition


Creative Commons License

# An example

Existential restriction on primitive class **Shark:**

necessarily hasMouthPart some **Teeth**

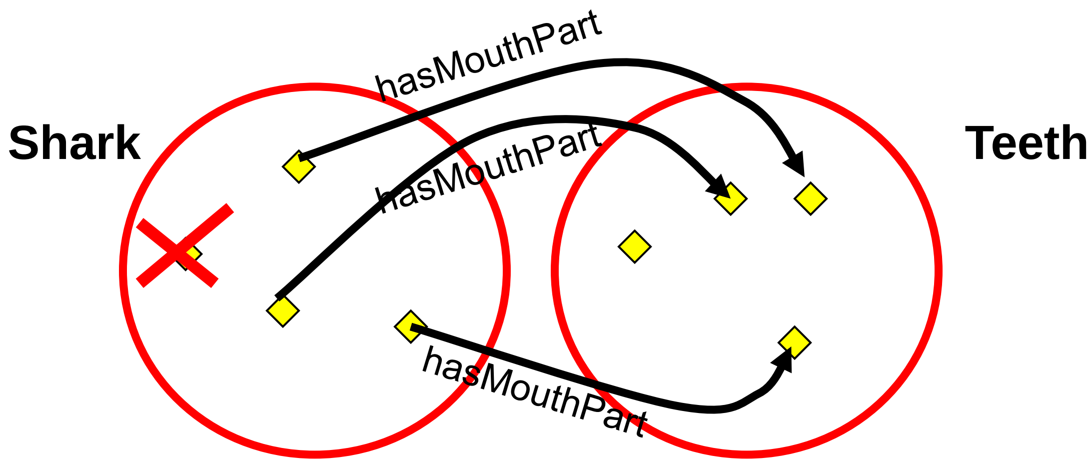



"Every member of the **Shark** class must have at least one mouthpart from the class **Teeth**"

# An example

Existential restriction on primitive class **Shark**:

necessarily hasMouthPart some **Teeth**




"There can be no member of **Shark**, that does not have at least one hasMouthPart relationship with an member of class **Teeth**"


Creative Commons License

# Restriction Types

| | |
|---|---|
| Existential, someValuesFrom | "Some", "At least one" |
| Universal, allValuesFrom | "Only" |
| hasValue | "equals x" |
| Cardinality | "Exactly n" |
| Max Cardinality | "At most n" |
| Min Cardinality | "At least n" |


Creative Commons License

# Exercise 4: Restrictions

- Create a restriction for pizzas stating that pizzas have some topping and have some base
- We demo this


Creative Commons License

# Exercise 5: Define some Named Pizzas

- Create a subclass of Pizza, called NamedPizza, and a subclass of NamedPizza, called  MargheritaPizza.
- Add an anonymous superclass for MargerithaPizza stating that MargerithaPizza has some MorzarellaTopping and some TomatoTopping
- In addition, to this example, create different kinds of pizza using the Pizza menu.
- We demo this


Creative Commons License

# CheesyPizza

- A CheesyPizza is any pizza that has some cheese on it
- We would expect then, that some pizzas might be named pizzas and cheesy pizzas (among other things later on)
- We can use the reasoner to help us produce this polyhierarchy without having to assert multiple parents


Creative Commons License

# Creating a CheesyPizza

- We normally create primitive classes and then migrate them to defined classes
- All of our defined pizzas will be direct subclasses of Pizza
- So, we create a CheesyPizza Class (do not make it disjoint) and add a restriction:
  "Every **CheesyPizza** must have at least one **CheeseTopping**"
  in the Superclasses widget
- Classifying shows that we currently don't have enough information to do any classification
- We then move the conditions from the *Superclasses* block to the *Equivalent classes* block which changes the meaning
- And classify again…



Creative Commons License

# Exercise 6: Create a Defined Class

- Add a class CheesyPizza below Pizza
- Add an anonymous superclass “hasTopping some CheeseTopping”
- Classify and look at the inferred hierarchy
- Add the anonymous class under Equivalent classes
- Classify again and check the inferred hierarchy


Creative Commons License

# Reasoner Classification

- The reasoner has been able to infer that anything that is a **Pizza** that has at least one topping from **CheeseTopping** is a **CheesyPizza**
- **MargheritaPizza** can be found under both **NamedPizza** and **CheeseyPizza** in the inferred hierarchy
- We don't currently have many kinds of primitive pizza but its easy to see that if we had, it would have been a substantial task to assert **CheesyPizza** as a parent of lots, if not all, of them
- And then do it all over again for other defined classes like **MeatyPizza** or whatever

Mission Successful!

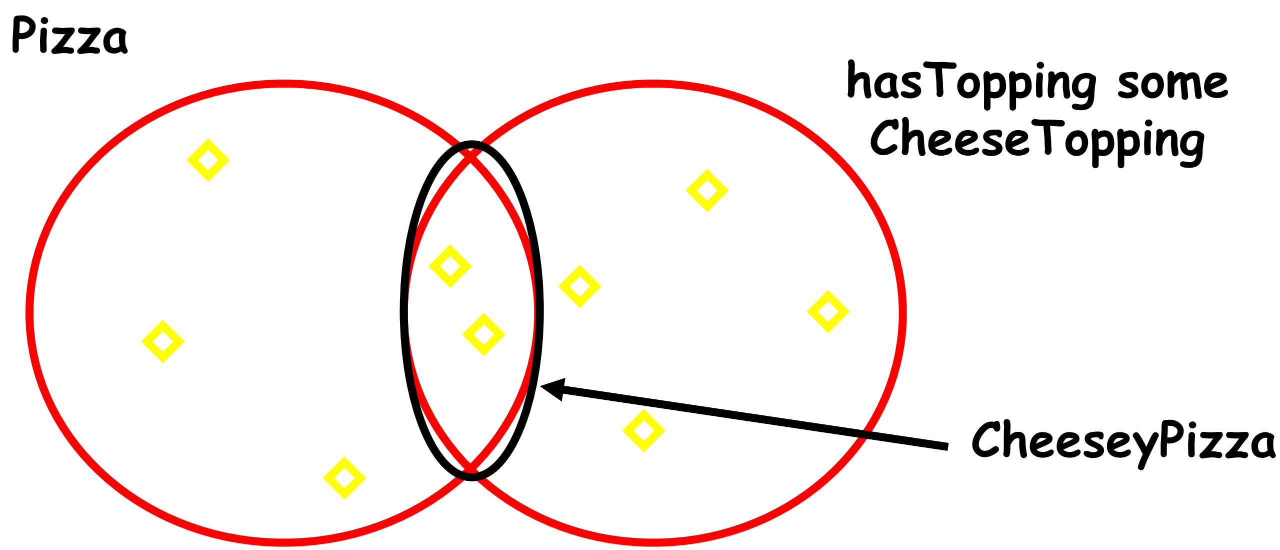

Creative Commons License

MANCHESTER 1824

The University of Manchester

# Why?
## Defined Classes

- Each set of necessary & sufficient conditions is an Equivalent Class




- **CheeseyPizza** is equivalent to the intersection of **Pizza** and **hasTopping some CheeseTopping**
- Classes, all of whose individuals fit this definition are found to be subclasses of **CheeseyPizza**, or are subsumed by **CheeseyPizza**

The University of Manchester
Creative Commons License

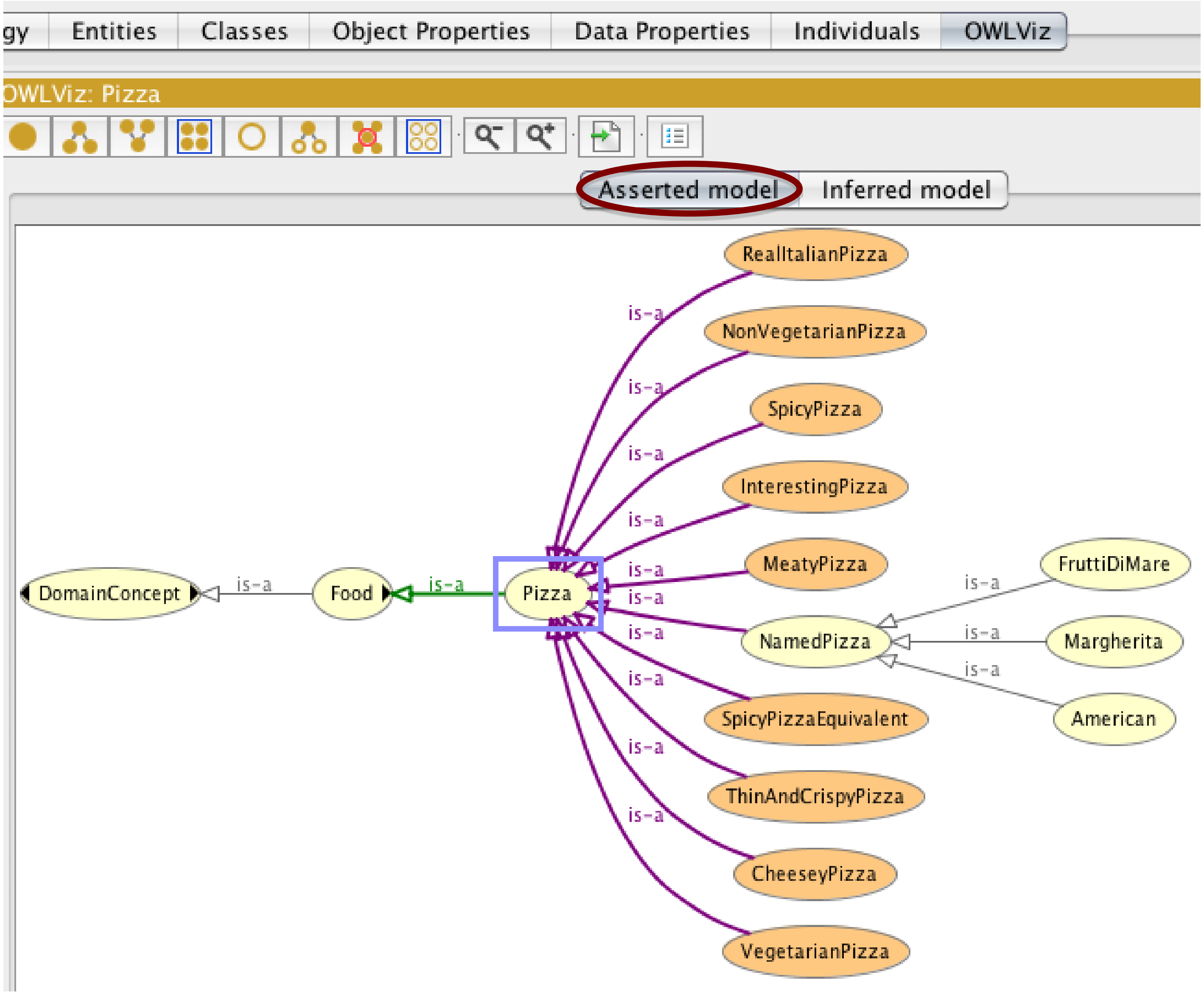
# Viewing polyhierarchies

- As we now have multiple inheritance, the tree view is less than helpful in viewing our “hierarchy”



Creative Commons License

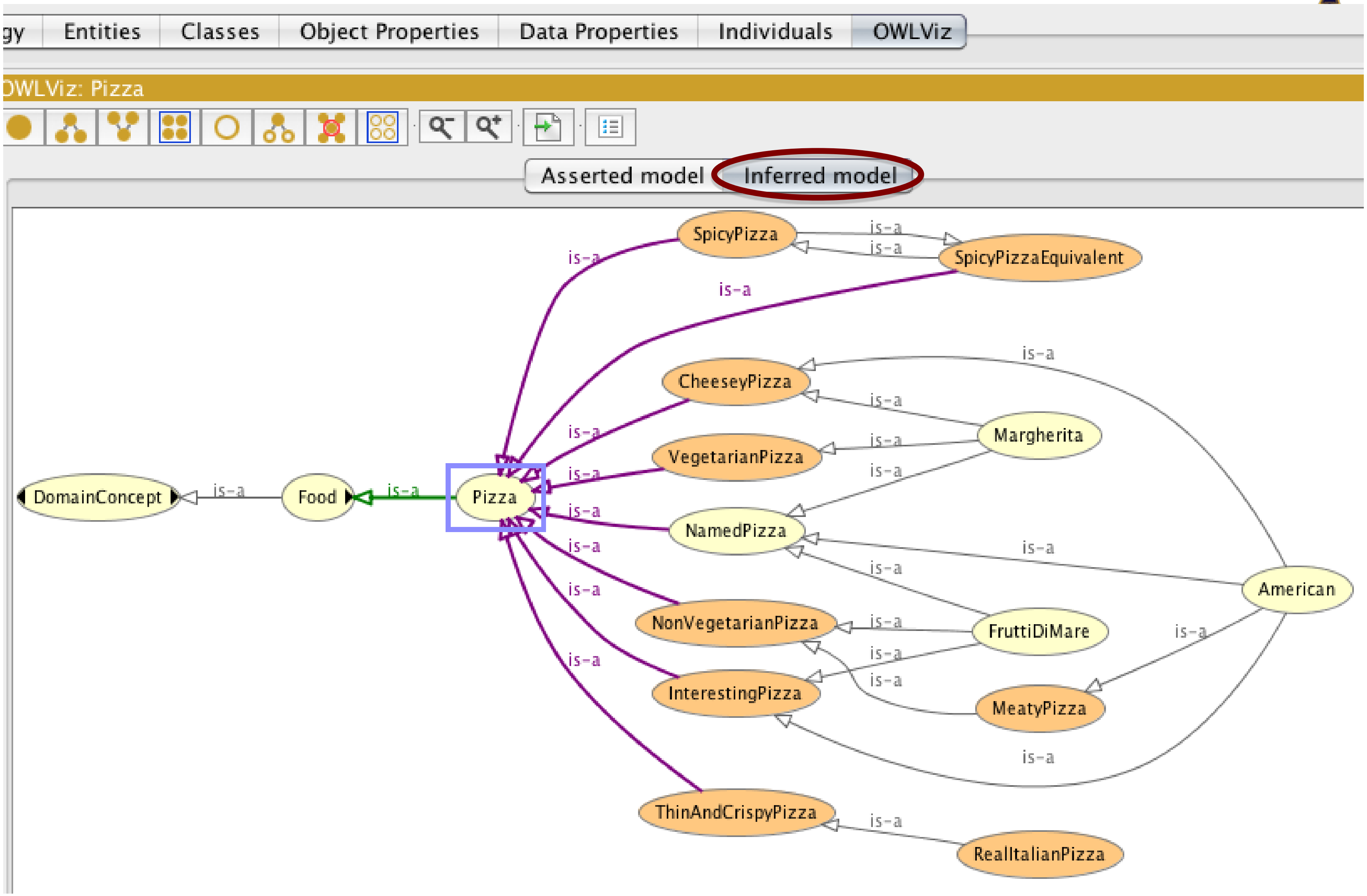
# Viewing our Hierarchy Graphically





Creative Commons License

# Viewing our Hierarchy Graphically





Creative Commons License

# Using OWLViz to untangle

- The asserted hierarchy should, ideally, be a tidy tree of disjoint primitives
- The inferred hierarchy will be tangled
- By switching from the asserted to the inferred hierarchy, it is easy to see the changes made by the reasoner
- OWLViz can be used to spot tangles in the primitive tree
- http://code.google.com/p/co-ode-owl-plugins/wiki/OWLViz

The University of Manchester
Creative Commons License

# Defined Classes

- We've created a Defined Class, **CheesyPizza**
  - It has a definition. That is *at least one* Necessary and Sufficient condition
  - Classes, all of whose individuals satisfy this definition, can be inferred to be subclasses
  - Therefore, we can use it like a query to "collect" subclasses that satisfy its conditions
  - Reasoners can be used to organise the complexity of our hierarchy
- It's marked with an equivalence symbol in the interface
- Defined classes are rarely disjoint


Creative Commons License

# Define a Vegetarian Pizza

- Not as easy as it looks…
- Define in words?
  - "a pizza with only vegetarian toppings"?
  - "a pizza with no meat (or fish) toppings"?
  - "a pizza that is not a MeatyPizza"?
- More than one way to model this

We'll start with the first example


Creative Commons License

# Define a Vegetarian Pizza

To be able to define a vegetarian pizza as

a Pizza with only Vegetarian Toppings

we need:

1. To be able to create a vegetarian topping
   This requires a Union Class

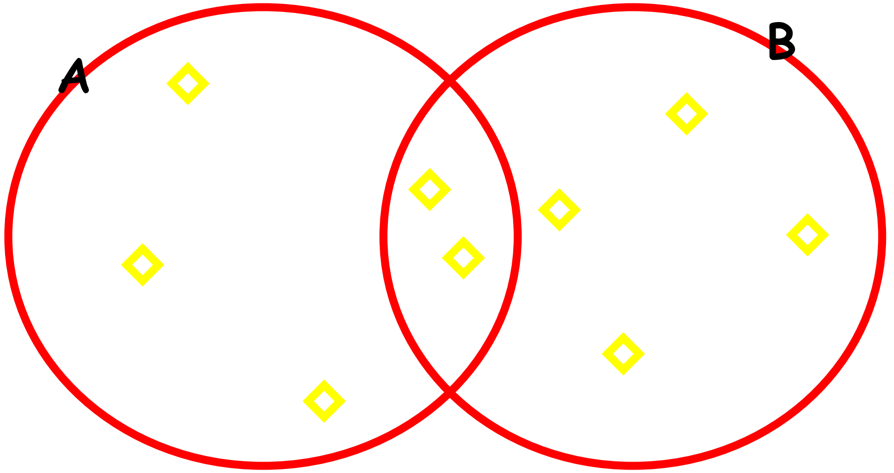
2. To be able to say "only"
   This requires a Universal Restriction


Creative Commons License

# Union Classes

- aka “disjunction”
- This OR That OR TheOther

**A or B** includes all individuals of class A and all individuals from class B and all individuals in the overlap (if A and B are not disjoint)




- Commonly used for:
  - Covering axioms
  - Closure


Creative Commons License

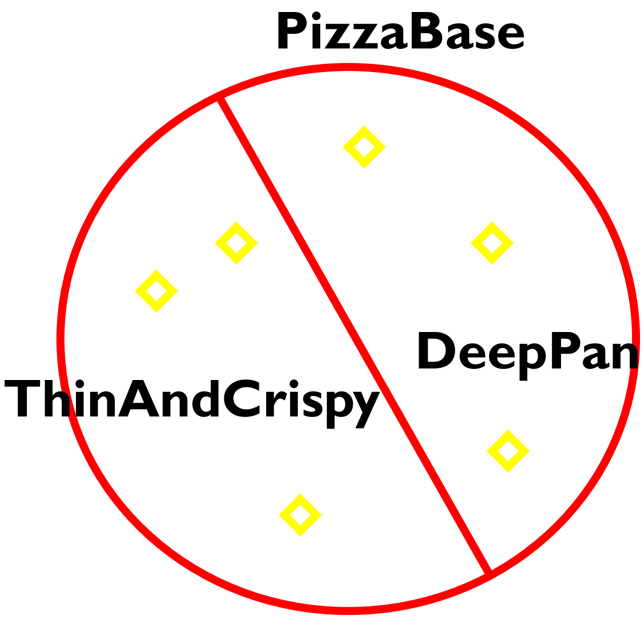
# Covering Axioms

- Covering axiom – a union expression containing several covering classes
- A covering axiom in the *Necessary & Sufficient* Conditions of a class means:
  the class cannot contain any instances other than those from the covering classes
- NB. If the covering classes are subclasses of the covered class, the covering axiom only needs to be a Necessary condition – it doesn't harm to make it Necessary & Sufficient though – its just redundant


Creative Commons License

# Covering PizzaBase

**PizzaBase ≡ ThinAndCrispy** or **DeepPan**




- In this example, the class **PizzaBase** is covered by **ThinAndCrispy** or **DeepPan**
- "All **PizzaBase**s must be **ThinAndCrispy** or **DeepPan**"
- "There are no other types of **PizzaBase**"

# Exercise 7: Define a Class VegetarianTopping


Creative Commons License

# Universal Restrictions

- We need to say our **VegetarianPizza** can only have toppings that are vegetarian toppings
- We can do this by creating a Universal or only restriction


Creative Commons License

# Exercise 8: Create a class VegetarianPizza


Creative Commons License

# VegetarianPizza Classification

- Nothing classifies under **VegetarianPizza**

- Actually, there is nothing wrong with our definition of **VegetarianPizza**
- It is actually the descriptions of our **Pizza**s that are incomplete

- The reasoner has not got enough information to infer that any **Pizza** is subsumed by **VegetarianPizza**

- This is because OWL makes the Open World Assumption


Creative Commons License

# Open World Assumption

- In a closed world (like DBs), the information we have is everything
- In an open world, we assume there is always more information than is stated

- Where a database, for example, returns a negative if it cannot find some data, the reasoner makes no assumption about the completeness of the information it is given
- The reasoner cannot determine something does not hold unless it is explicitly stated in the model


Creative Commons License

# Open World Assumption

- Typically we have a pattern of several Existential restrictions on a single property with different fillers – like primitive pizzas on hasTopping
- Existential restrictions should be paraphrased by "amongst other things…"
- Must state that a description is complete
- We need closure for the given property


Creative Commons License

# Closure

- This is in the form of a Universal Restriction with a filler that is the Union of the other fillers for that property

- Closure works along a single property


Creative Commons License

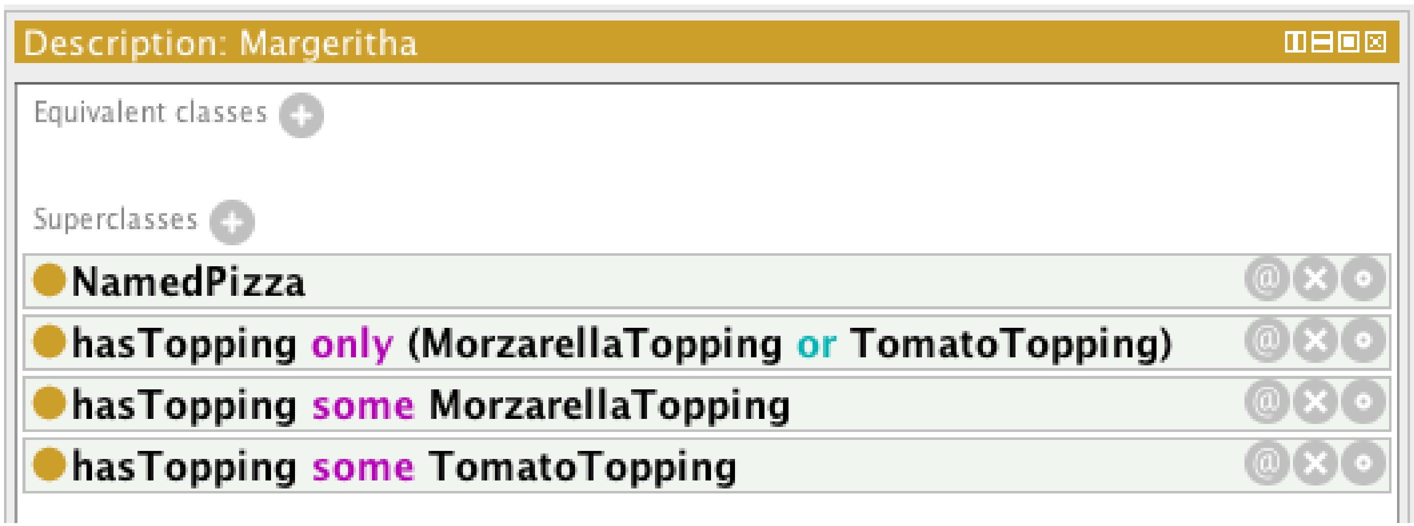
# Closure example: MargheritaPizza

All **MargheritaPizzas** must have:

at least 1 topping from **MozzarellaTopping** and

at least 1 topping from **TomatoTopping**

only toppings from **MozzarellaTopping** or **TomatoTopping**




- The last part is paraphrased into “no other toppings”
- The union closes the hasTopping property on **MargheritaPizza**



Creative Commons License

# Exercise 9: Closing Pizzas


Creative Commons License

# Summary

You should now be able to:

- extract Knowledge (and act as an expert)
- identify components of the Protégé-OWL Interface
- create Primitive Classes and Properties
- create some basic Restrictions on a Class
- Create Defined Classes and classify using a reasoner to check expected results
- Create Covering Axioms
- Close Class Descriptions and understand the Open World Assumption


Creative Commons License

# Reference Material

- Further material is available from: http://owl.cs.manchester.ac.uk/tutorials/protegeowltutorial/
- Protégé: http://protege.stanford.edu/
  Protégé wiki: http://protegewiki.stanford.edu/
- HermiT OWL Reasoner: http://www.hermit-reasoner.com
- Pellet OWL Reasoner: http://clarkparsia.com/pellet/
- OWLViz: http://protegewiki.stanford.edu/wiki/OWLViz
- Pizza Finder: http://www.co-ode.org/downloads/pizzafinder/


Creative Commons License

# Extra Exercise 10: Cardinality Restrictions

- In OWL we can describe the class of individuals that have *at least*, *at most* or *exactly a specific number of relationships with other individuals or datatype values.*
- We have min, max and exactly Cardinality Restrictions.
- We can create **InterestingPizza**, which is defined a a Pizza that has at least 3 PizzaToppings.

# Extra Exercise 11: Qualified Cardinality Restrictions

- QCRs are more specific than the previous example in that they state the class of objects within the restriction
- We can define a type of FourCheesePizza, that is defined as having exactly four cheese toppings.
- Can a four cheese pizza have other toppings other than cheese?


Creative Commons License

# Building Ontology-based Applications using Pellet

International Semantic Web Conference 2010

*Bernardo Cuenca-Grau*
*Oxford University Computing Laboratory*

# What is Clark & Parsia?

- Small semantic software firm in Washington, DC and Boston
- Provides software development and integration services
- Specializing in Semantic Web, web services, and advanced AI technologies for federal and enterprise customers

http://clarkparsia.com/
Twitter: @candp

# What is Pellet?

- Pellet is an OWL-DL reasoner
  - Supports OWL 2
  - Sound and complete reasoner
- Written in Java and available from http://clarkparsia.com/pellet
- Dual-licensed
  - AGPL license for open-source applications
  - Commercial license for commercial applications

# Talk Roadmap

- OWL and Reasoning
- Developing ontologies
  - Validate and debug schema definitions
- Connecting multiple ontologies
  - Ontology alignment
- Validating instance data
  - Identify and resolve inconsistencies in the data
- Reasoning with instance data
  - Answer queries over combined data using Pellet
  - Scalability and performance considerations

# OWL and Reasoning

# OWL in 3 Slides (1)

## ENTITIES

- Class: Person, Organization, Project, Skill, ...
- Datatype: string, integer, date, ...

- Individual: Evren, C&P, POPS, ...
- Literal: "Evren Sirin", 5, 5/26/2008, ...

- Object Property: worksAt, hasSkill, ...
- Data property: name, proficiencyLevel, ...

# OWL in 3 Slides (2)

## EXPRESSIONS

- Class expressions
  - and, or, not
  - some, only, min, max, exactly, value, Self
  - { ... }
- Datatype definitions
  - and, or, not
  - <, <=, >, >=
  - { ... }

# OWL in 3 Slides (3)

## AXIOMS

- Class axioms
  - subClassOf, equivalentTo, disjointWith
- Property axioms
  - subPropertyOf, equivalentTo, inverseOf, disjointWith, subPropertyChain, domain, range
- Property characteristics
  - Functional, InverseFunctional, Transitive, Symmetric, Asymmetric, Reflexive, Irreflexive
- Individual assertions
  - Class assertion, property assertion, sameAs, differentFrom

# OWL Example

- Employee **equivalentTo** ( CivilServant **or** Contractor )
- CivilServant **disjointWith** Contractor
- Employee **subClassOf**
  employeeID **some** integer[>= 100000, <= 999999]
- Employee **subClassOf** employeeID **exactly** 1
- worksOnProject **domain** Person
- worksOnProject **range** Project
- Person0853 **type** CivilServant
- Person0853 employeeID 312987
- Person0853 worksOnProject Project2133

# OWL Example

- Employee **equivalentTo** ( CivilServant **or** Contractor )
- CivilServant **disjointWith** Contractor
- Employee **subClassOf**
  employeeID **some** integer[>= 100000, <= 999999]
- Employee **subClassOf** employeeID **exactly** 1
- worksOnProject **domain** Person
- worksOnProject **range** Project

Schema (TBox)

- Person0853 **type** CivilServant
- Person0853 employeeID 312987
- Person0853 worksOnProject Project2133

Data (ABox)

# Reasoning in OWL

1. Check the consistency of a set of axioms
   - Verify the input axioms do not contain contradictions

# Inconsistency Examples

- Example 1
  - CivilServant **disjointWith** Contractor
  - Person0853 **type** CivilServant , Contractor
- Example 2
  - ActiveProject **subClassOf** endDate **max** 0
  - Project2133 **type** ActiveProject
  - Project2133 endDate "1/1/2008"^^xsd:date

# Unsatisfiability

- Unsatisfiable class cannot have any instances
  - Consistent ontologies may contain unsatisfiable classes
  - Declaring an instance for an unsatisfiable class causes inconsistency
- Example
  - CivilServant **disjointWith** Contractor
  - CivilServantContractor **subClassOf** ( CivilServant **and** Contractor )

# Reasoning in OWL

1. Check the consistency of a set of axioms
   - Verify the input axioms do not contain contradictions
   - ***Mandatory first step before any other reasoning service***
   - Fix the inconsistency before reasoning
     - Why?
     - Because ***any consequence*** can be inferred from inconsistency

# Inference Examples

- Input axioms
  1. Employee **equivalentTo** ( CivilServant **or** Contractor )
  2. CivilServant **disjointWith** Contractor
  3. isEmployeeOf **inverseOf** hasEmployee
  4. isEmployeeOf **domain** Employee
  5. Person0853 **type** CivilServant
  6. Person0853 isEmployeeOf Organization5349
- Some inferences
  - CivilServant **subClassOf** Employee { 1 }
  - Person0853 **type** Employee { 1, 5 }, { 4, 6 }
  - Person0853 **type** **not** Contractor { 2, 5 }
  - Organization5349 hasEmployee Person0853 { 3, 6 }

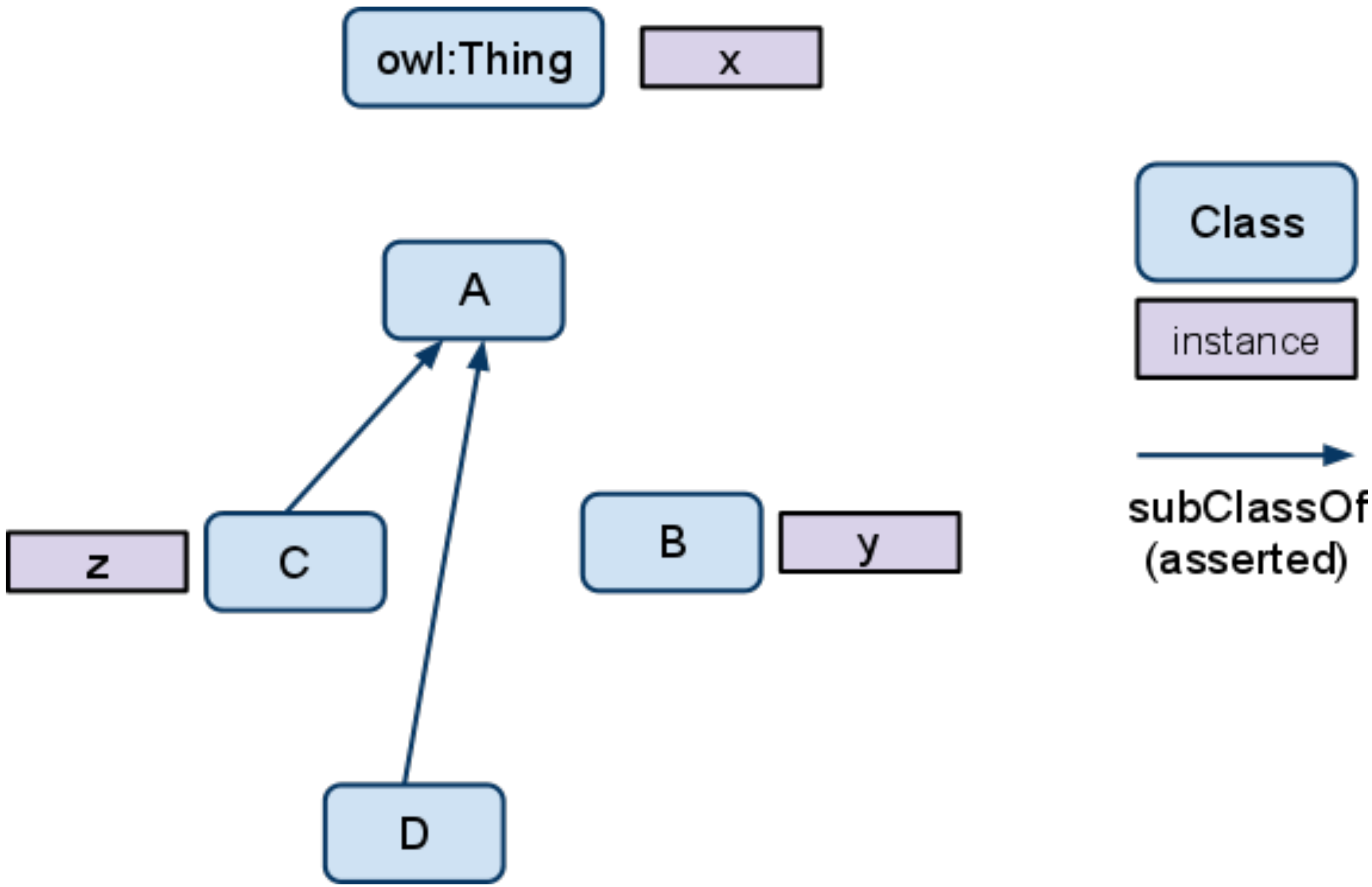
# Reasoning in OWL

1. Check the consistency of a set of axioms
    - Verify the input axioms do not contain contradictions
    - Mandatory first step before any other reasoning service
    - Fix the inconsistency before reasoning
        - Any consequence can be inferred from inconsistency
2. Infer new axioms from a set of axioms
    - Truth of an axiom is logically proven from asserted axioms
    - Infinitely many inferences for any non-empty ontology
    - Inferences can be computed as a batch process or as required by queries

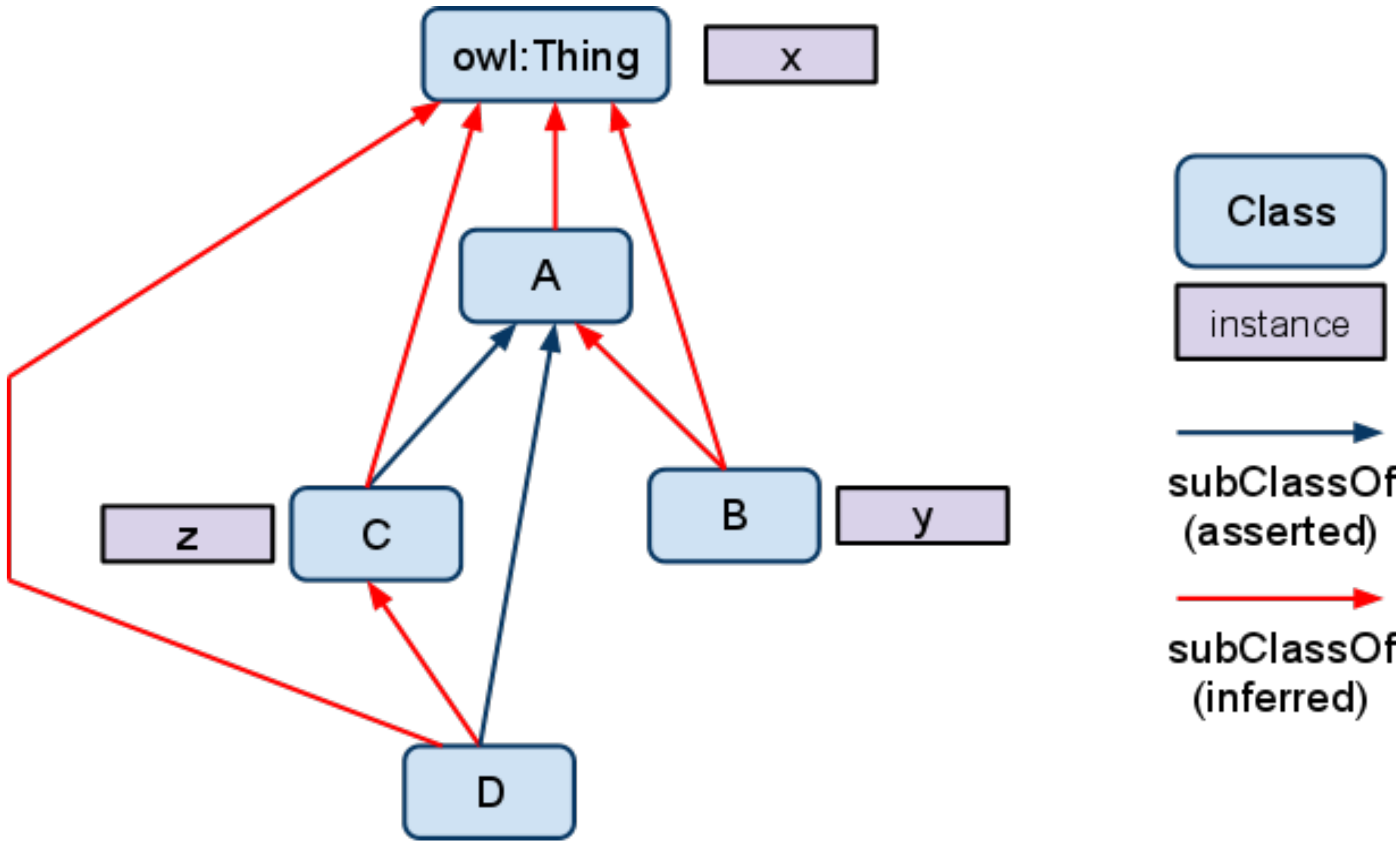
# Common Reasoning Tasks

- Classification
  - Compute subClassOf and equivalentClass inferences between all named classes
- Realization
  - Find most specific types for each instance
  - Requires classification to be performed first

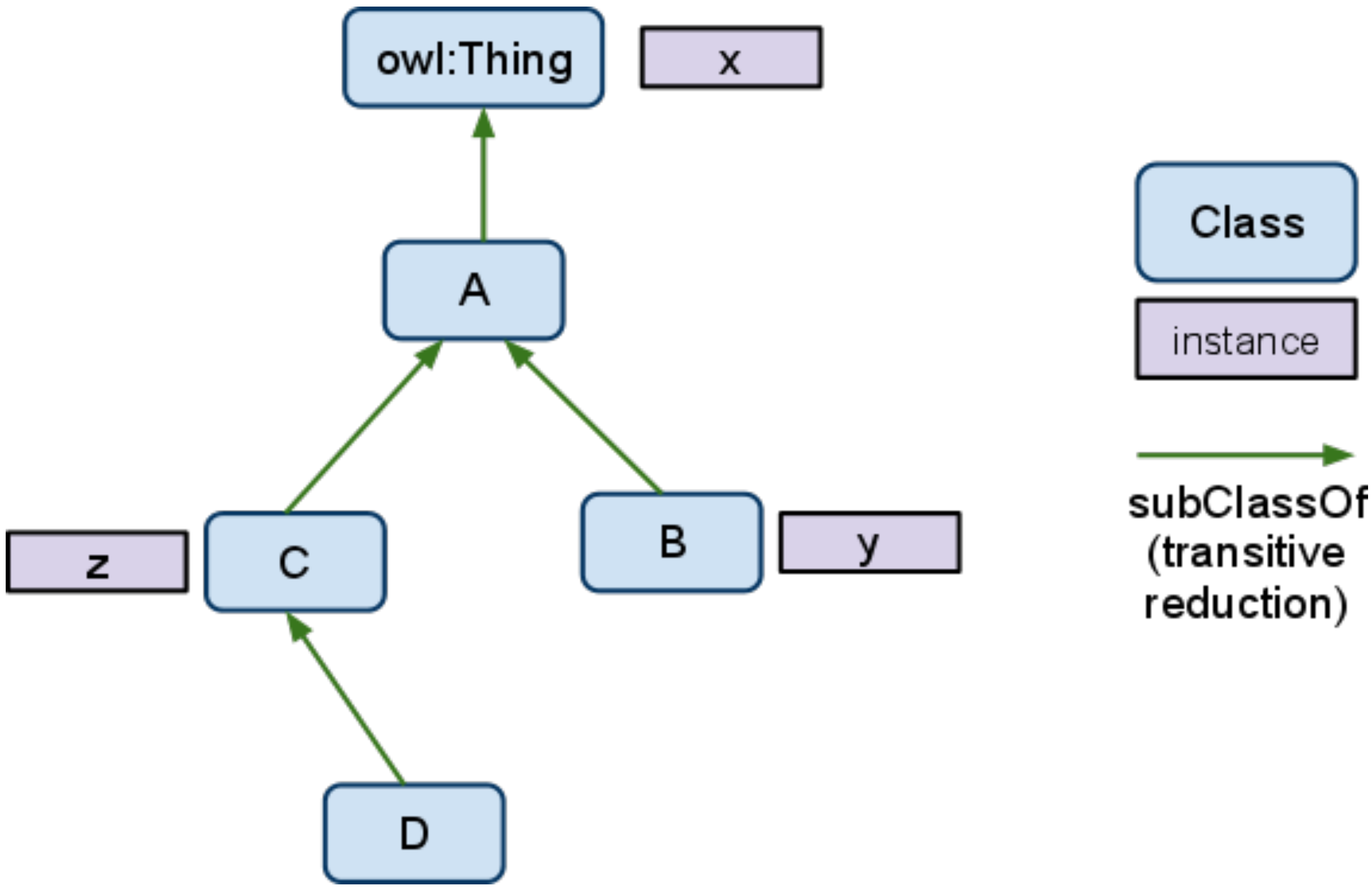
# Asserted Ontology



# Inferred Subclasses



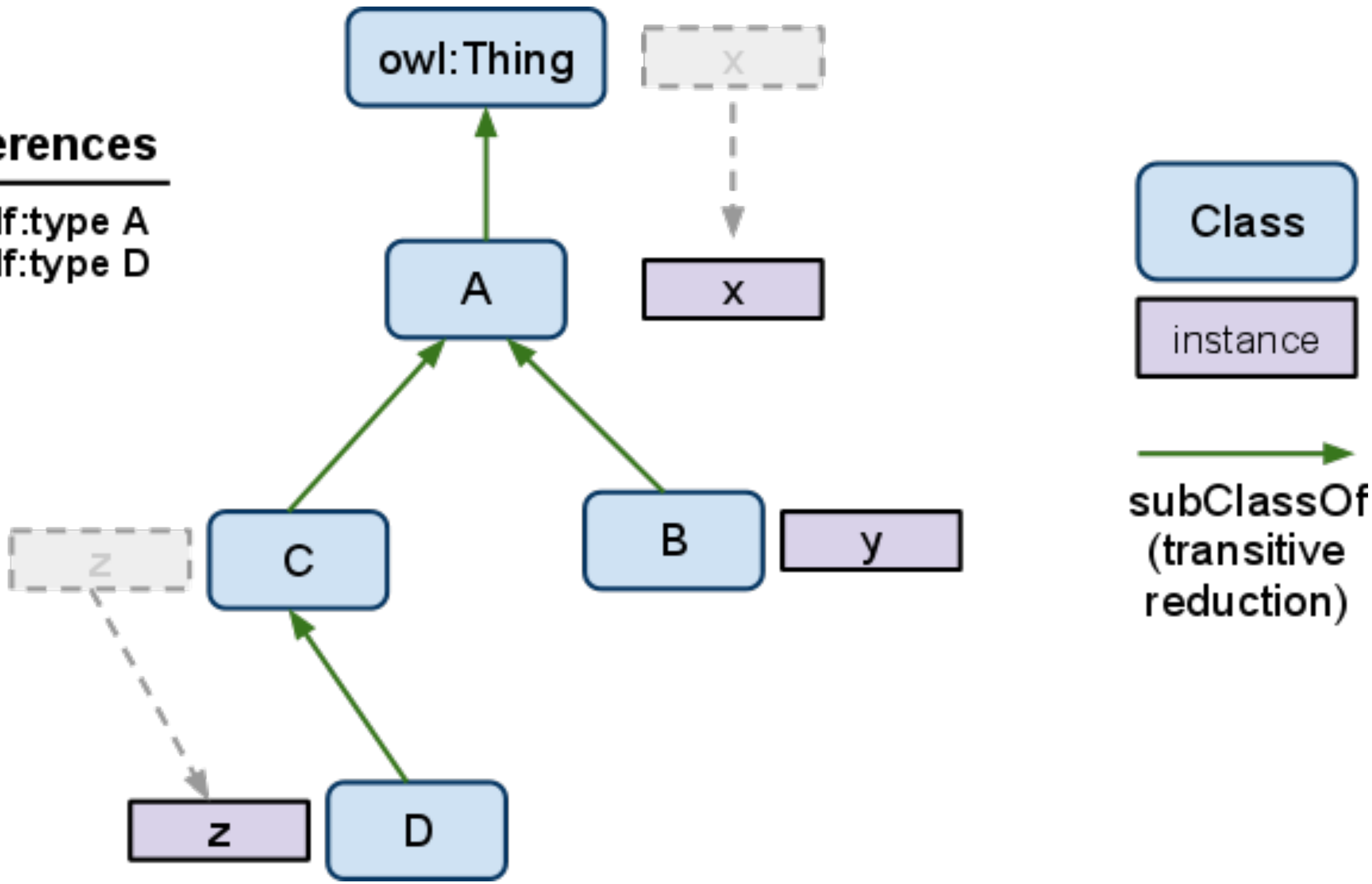
# Classification Tree



# Instance Realization

**Inferences**

x rdf:type A
z rdf:type D



# SPARQL Queries

- Retrieve subclasses

```
SELECT ?C WHERE {
    ?C rdfs:subClassOf :Employee .
}
```

- Retrieve instances

```
SELECT ?X WHERE {
    ?X rdf:type :Employee .
}
```

- Retrieve subclasses and their instances

```
SELECT ?X ?C WHERE {
    ?X rdf:type ?C .
    ?C rdfs:subClassOf :Employee .
}
```

# Ontology Development

# Developing Ontologies

- Incremental, iterative process
  - We do not have to start with a perfect model!
- Derive the ontology from the sources
- Link it to existing ontologies!
- Revise, modify, improve
  - Editing tools
    - Protégé
    - TopBraid composer
  - Use reasoning to get it right!

# Building the Ontology from the Sources

- Structured data
  - Map tables and columns to concepts and relations

```
SELECT id, name, address
FROM people
WHERE age >= 18
```

→

```
Adult(id),
hasName(id, name),
livesIn(id, address)
```

- Class: Adult
- Relations: hasName, livesIn

# Does the Data change frequently?

- No
  - Extract, Transform, Load
  - Execute queries over sources
  - Populate concepts and relations
- Yes
  - Query rewriting
  - Leave the data where it is
  - Use the mappings when querying the ontology

# Dealing with Semi/Unstructured Data

- Extract metadata from documents
  - Filetype, Author, Description, ...
- Extract further knowledge from the content
  - Keywords
  - Topics
  - Key sentences
  - Document similarity
  - Coreference resolution
  - Concepts
  - Relations

# Ontology Alignment

# POPS and FOAF

- People, Organizations, Projects, and Skills ontology
  - Developed by Clark & Parsia
  - Expertise location in a large organization (NASA)
  - People, contact information, work history, evidence of skills, publications, etc.
- Friend Of A Friend ontology
  - Project devoted to linking people and information using the Web
  - People, agents, projects, organizations, etc.

# Data Integration

- Integrate data from multiple sources
- Sources use different vocabularies
- Establish a common vocabulary to enable uniform access to all data sources
    - Use single queries to retrieve instances from all relevant data sets

# Simple Alignment

- pops:Employee **subClassOf** foaf:Person
- pops:Project **equivalentTo** foaf:Project
- pops:Organization **equivalentTo** foaf:Organization
- pops:hasEmployee **subPropertyOf** foaf:member
- pops:mbox_sha1sum **equivalentTo** foaf:mbox_sha1sum

# Alignment with SWRL

- Mapping sometimes not straight-forward
  - POPS defines firstName and lastName
  - FOAF defines name
  - Concat first and last names to get the full name
- SWRL rule with a built-in function

```
pops:firstName(?person, ?first) ^
pops:lastName(?person, ?last) ^
?name = swrlb:concat(?first " " ?last)
=>
foaf:name(?person, ?name)
```

# More SWRL Mapping

- Another example
  - POPS uses worksOnProject property for both current and previous projects
  - FOAF distinguishes currentProject and pastProject
- Solution: POPS also defines ActiveProject class
- SWRL rule to encode conditional subproperty

pops:worksOnProject(?person, ?project) ^
pops:ActiveProject(?project)
=>
foaf:currentProject(?person, ?project)

# Programming with Pellet

# APIs for accessing Pellet

- Pellet can be used via four different APIs
  - Internal Pellet API (Deprecated soon...)
  - New (2.3) native Pellet API: Ortiz
  - Manchester OWLAPI
  - Jena
- Each API has pros and cons
  - Choice will depend on your applications' needs and requirements

# Pellet Internal API

- API used by the reasoner
  - Designed for efficiency, not usability
  - Uses ATerm library for representing terms
  - Fine-grained control over reasoning
  - Misses features (e.g. parsing & serialization)
- Pros: Efficiency, fine-grained control
- Cons: Low usability, missing features
- **Big Con: Will be deprecated in Pellet 2.3**

# Ortiz API

- New API designed for OWL
  - idiomatic, Java-friendly API
  - one API for *the Pellet family of OWL 2 reasoners*
  - not slavishly tied to OWL 2 specifications
  - unifies:
    - SPARQL queries
    - SWRL rules
    - OWL axioms
- Pros: Very Java-friendly, OWL-centric API
- Cons: New...

# Manchester OWLAPI

- API designed for OWL
  - Closely tied to OWL structural specification
  - Support for many syntaxes (RDF/XML, OWL/XML, OWL functional, Turtle, ...)
  - Native SWRL support
  - Integration with reasoners
  - Support for modularity and explanations
- Pros: OWL-centric API
- Cons: Not as stable, no SPARQL support (yet)
- More info: http://owlapi.sf.net

# Jena API

- RDF framework developed by HP labs
  - An RDF API with OWL extensions
  - In-memory and persistent storage
  - Built-in rule reasoners and integrated with Pellet
  - SPARQL query engine
- Pros: Mature and stable and ubiquitous
- Cons: Not great for handling OWL, no specific OWL 2 support
- More info: http://jena.sf.net

# Jena Basics

- **Model** contains set of **Statements**
- **Statement** is a triple where
  - Subject is a **Resource**
  - Predicate is a **Property**
  - Object is an **RDFNode**
- **InfModel** extends **Model** with inference
- **OntModel** extends **InfModel** with ontology API

# Creating Inference Models

```
// create an empty non-inferencing model
Model rawModel = ModelFactory.createDefaultModel();

// create Pellet reasoner
Reasoner r = PelletReasonerFactory.theInstance().create();

// create an inferencing model using the raw model
InfModel model = ModelFactory.createInfModel(r, rawModel);
```

# Creating Ontology Models

```
// create an empty non-inferencing model
Model rawModel = ModelFactory.createDefaultModel();

// create an ontology model using Pellet spec and raw model
OntModel model = ModelFactory.createOntologyModel(
      PelletReasonerFactory.THE_SPEC, rawModel);
```

# Which Model to Use?

- Ontology API may introduce some overhead
  - Additional object conversions (from RDF API objects to OWL API objects)
  - Additional queries to the underlying reasoner

# Data Validation

# Consistency Checking

```
// create an inferencing model using Pellet reasoner
InfModel model = ModelFactory.createInfModel(r, rawModel);

// get the underlying Pellet graph
PelletInfGraph pellet = (PelletInfGraph) model.getGraph();

// check for inconsistency
boolean consistent = pellet.isConsistent();
```

# Explaining Inconsistency

```
// IMPORTANT: The option to enable tracing should be turned
// on before the ontology is loaded to the reasoner!
PelletOptions.USE_TRACING = true;

// create an inferencing model using Pellet reasoner
InfModel model = ModelFactory.createInfModel(r, rawModel);
PelletInfGraph pellet = (PelletInfGraph) model.getGraph();

// create an inferencing model using Pellet reasoner
if( !pellet.isConsistent() ) {
   // create an inferencing model using Pellet reasoner
   Model explanation = pellet.explainInconsistency();
   // print the explanation
   explanation.write( System.out );
}
```

# Dealing with Inconsistency

- Inconsistencies are unavoidable
  - Distributed data, no single point of enforcement
  - Expressive modeling language
- Classical logical formalisms are not good at dealing with inconsistency
  - Reasoners refuse to reason with inconsistent ontologies
- Paraconsistent logics not practical
  - Complexity, tool support, etc.
- What can we do?

# An Automated Solution

- Typical process for solving a contradiction
  - Use Pellet to find which axioms cause contradiction
  - Domain expert (human) inspects the axiom set
  - Expert edits/deleted incorrect axioms
- An automated (and cautious) solution
  - Use Pellet to find which axioms cause contradiction
  - Delete all reported axioms (WIDTIO)
- When to use the automated solution
  - Pros: Completely automated, guaranteed to retain only consistent information
  - Cons: May remove too much information

# Resolving Inconsistencies

```
// continue until all inconsistencies are resolved
while (!pellet.isConsistent()) {
  // get the explanation for current inconsistency
  Graph explanation = pellet.explainInconsistency();
  // iterate over the axioms in the explanation
  for (Triple triple : explanation.find(Triple.ANY).toList() ) {
     // remove any individual assertion that contributes
     // to the inconsistency (assumption: all the axioms
     // in the schema are believed to be correct and
     // should not be removed)
     if (isIndividualAssertion(triple))
       graph.remove(triple);
  }
}
```

# Closed vs. Open World

- Two different views on truth
  - CWA: Any statement that is not known to be true is false
  - OWA: A statement is false only if it is known to be false
- Used in different contexts
  - Databases use CWA because (typically) you have *complete* information
  - Ontologies use OWA because (typically) you have *incomplete* information
- Data validation results significantly different when using CWA instead of OWA

# Example (1)

- Input axioms
  - Employee **subClassOf** employeeID **some** integer
  - Person0853 **type** Employee
- OWA
  - Consistent: true
  - Reason: Person0853 has an employeeID but we don't know the exact value
- CWA
  - Consistent: false
  - Reason: Person0853 does not have an employeeID

# Example (2)

- Input axioms
  - isEmployeeOf `range` Organization
  - Person0853 isEmployeeOf Organization5349
- OWA
  - Consistent: true
  - Inference: Organization5349 `type` Organization
- CWA
  - Consistent: false
  - Reason: Organization5349 `type` Organization is not explicitly asserted

# Example (3)

- Input axioms
  - hasManager **`Functional`**
  - Organization5349 hasManager Person0853
  - Organization5349 hasManager Person1735
- OWA
  - Consistent: true
  - Inference: Person0853 **`sameAs`** Person1735
- CWA
  - Consistent: false
  - Reason: Organization5349 has more than one value for hasManager

# CWA or OWA Validation?

- Should I use CWA or OWA?
  - Of course use both!
  - In the application domain there is complete information about some parts but not others
- We might have...
  - Complete knowledge about employees
  - Incomplete information about external publications
    - Retrieved from conference proceedings, etc
- An axiom can be interpreted with...
  - OWA - regular OWL axiom
  - CWA - integrity constraint (IC)

# How to use ICs in OWL

- Two easy steps
  1. Specify which axioms should be ICs
  2. Validate ICs with Pellet
- Ontology developer
  - Develop ontology as usual
  - Separate ICs from regular axioms
    - Annotation, separation of files, named graphs, ...
- Pellet IC validator
  - Translates ICs into SPARQL queries automatically
  - Execute SPARQL queries with Pellet
  - Query results show constraint violations
- Download: http://clarkparsia.com/pellet/download/oicv-0.1.1

# IC Validation

```
// create an inferencing model using Pellet reasoner
InfModel dataModel = ModelFactory.createInfModel(r);

// load the schema and instance data to Pellet
dataModel.read( "file:data.rdf" );
dataModel.read( "file:schema.owl" );

// Create the IC validator and associate it with the dataset
JenaICValidator validator = new JenaICValidator(dataModel);

// Load the constraints into the IC validator
validator.getConstraints().read("file:constraints.owl");

// Get the constraint violations
Iterator<ConstraintViolation> violations =
                                        validator.getViolations();
```

# Resolving IC Violations

- IC violations are similar to logical inconsistencies but not exactly the same
  - Lack of information may cause IC violation
- ICs do not cause new inferences
  - Used to detect violations
- Resolving IC violations
  - Add more information
    - Example: Add the missing employee ID info
  - Delete existing information
    - Example: Remove the employee

# Query Answering

# Querying via RDF API

```
// Get the resource we want to query about
Resource Employee = model.getResource(
        NS + "Employee" );
// Retrieve subclasses
Iterator subClasses = model.listSubjectsWithProperty(
        RDFS.subClassOf, Employee);
// Retrieve direct subclasses
Iterator directSubClasses = model.listSubjectsWithProperty(
        ReasonerVocabulary.directSubClassOf, Employee);
// Retrieve instances
Iterator instances = model.listSubjectsWithProperty(
        RDF.type, Employee);
```

# Querying via Ontology API

```
// Get the resource we want to query about
OntClass Employee = ontModel.getResource(
        NS + "Employee" );
// Retrieve subclasses
Iterator subClasses = Employee.listSubClasses();
// Retrieve direct subclasses
Iterator supClasses = Employee.listSubClasses(true);
// Retrieve instances
Iterator instances = Employee.listInstances();
```

# Querying with SPARQL

```
Query query = Query.create(
        PREFIXES +
        "SELECT ?X ?C " +
        "WHERE {" +
        "     ?X rdf:type ?C ." +
        "     ?C rdfs:subClassOf :Employee ." +
        "}" );
// Create a query execution engine with a Pellet model
QueryExecution qe =
             QueryExecutionFactory.create(query, model);

// Run the query
ResultSet results = qe.execSelect();
```

# ...with SPARQL-DL

```
Query query = Query.create(
        PREFIXES +
        "SELECT ?X ?C " +
        "WHERE {" +
        "     ?X sparqldl:directType ?C ." +
        "     ?C rdfs:subClassOf :Employee ." +
        "}" );
// Create a query execution engine with a Pellet model
QueryExecution qe =
      SparqlDLQueryExecutionFactory.create(query, model);

// Run the query
ResultSet results = qe.execSelect();
```

# SPARQL Engines

- ARQ query engine (comes with Jena)
  - ARQ handles the query execution
  - Calls Pellet with single triple queries
  - Supports all SPARQL constructs
  - Does not support OWL expressions
- Pellet query engine
  - Pellet handles the query execution
  - Supports only Basic Graph Patterns
  - Supports OWL expressions
- Mixed query engine
  - ARQ handles SPARQL algebra, Pellet handles Basic Graph Patterns
  - Supports all OWL and SPARQL constructs

clark&parsia

# Questions?

# More info

- Clark & Parsia, LLC
  - http://clarkparsia.com/
- News, updates, tips/tricks on twitter
  - #candp

# Thank you!