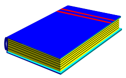
Ontology languages

Oscar Corcho
ocorcho@fi.upm.es
http://www.oeg-upm.net/

Ontological Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain

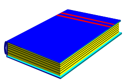
Main References

Gómez-Pérez, A.; Fernández-López, M.; Corcho, O. **Ontological Engineering.** *Springer Verlag. 2003*



Baader F, McGuinness D, Nardi D, Patel-Schneider P (2003)
The Description Logic Handbook: Theory, implementation and applications.
Cambridge University Press, Cambridge, United Kingdom

http://knowledgeweb.semanticweb.org



Research deliverables
Industry deliverables

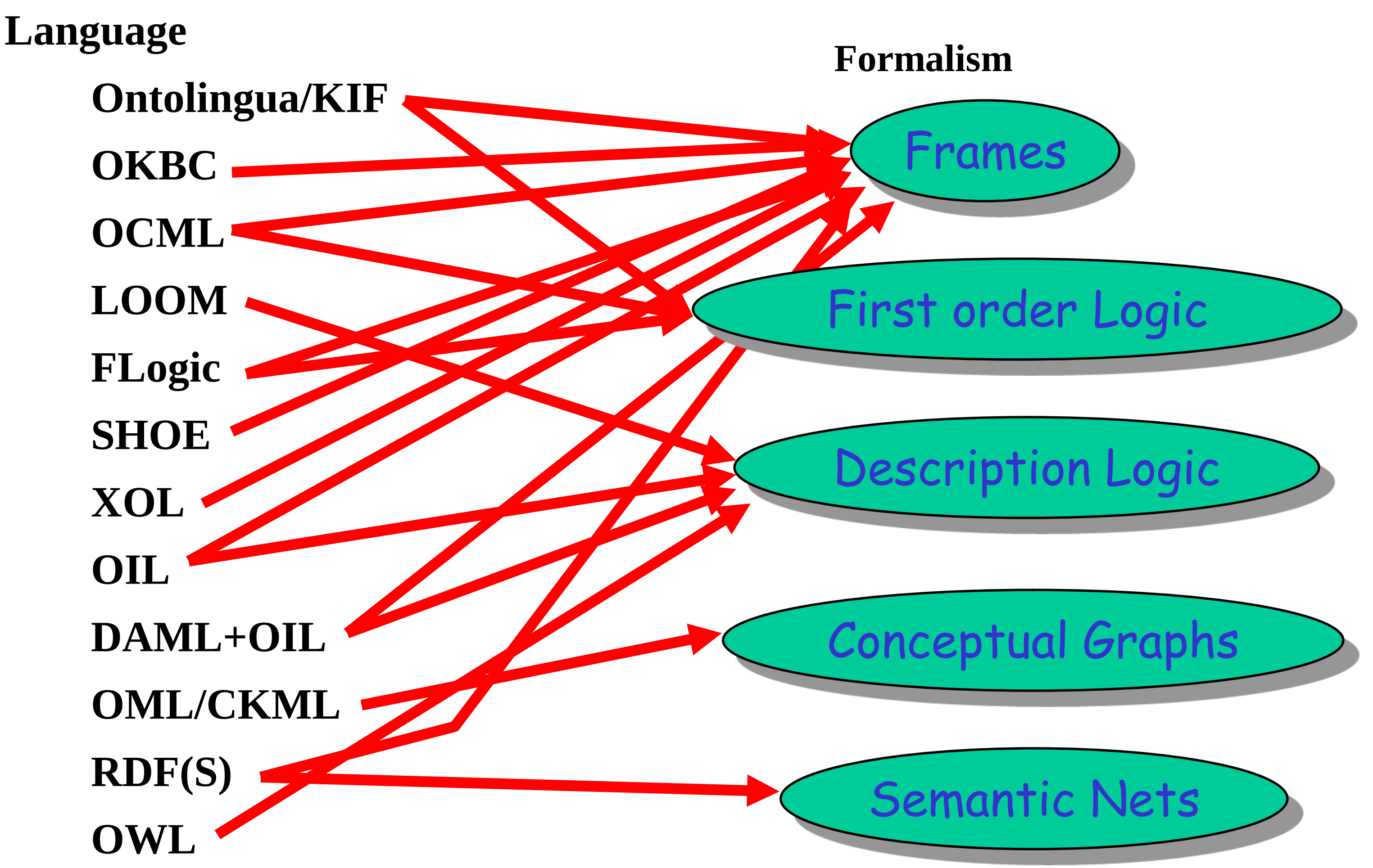
Dean M, Schreiber G (2004) *OWL Web Ontology Language Reference.* W3C Recommendation. *http://www.w3.org/TR/owl-ref/*
Brickley D, Guha RV (2004) *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. *http://www.w3.org/TR/PR-rdf-schema*
Lassila O, Swick R (1999) *Resource Description Framework (RDF) Model and Syntax Specification*. W3C Recommendation. *http://www.w3.org/TR/REC-rdf-syntax/*

Acknowledgements

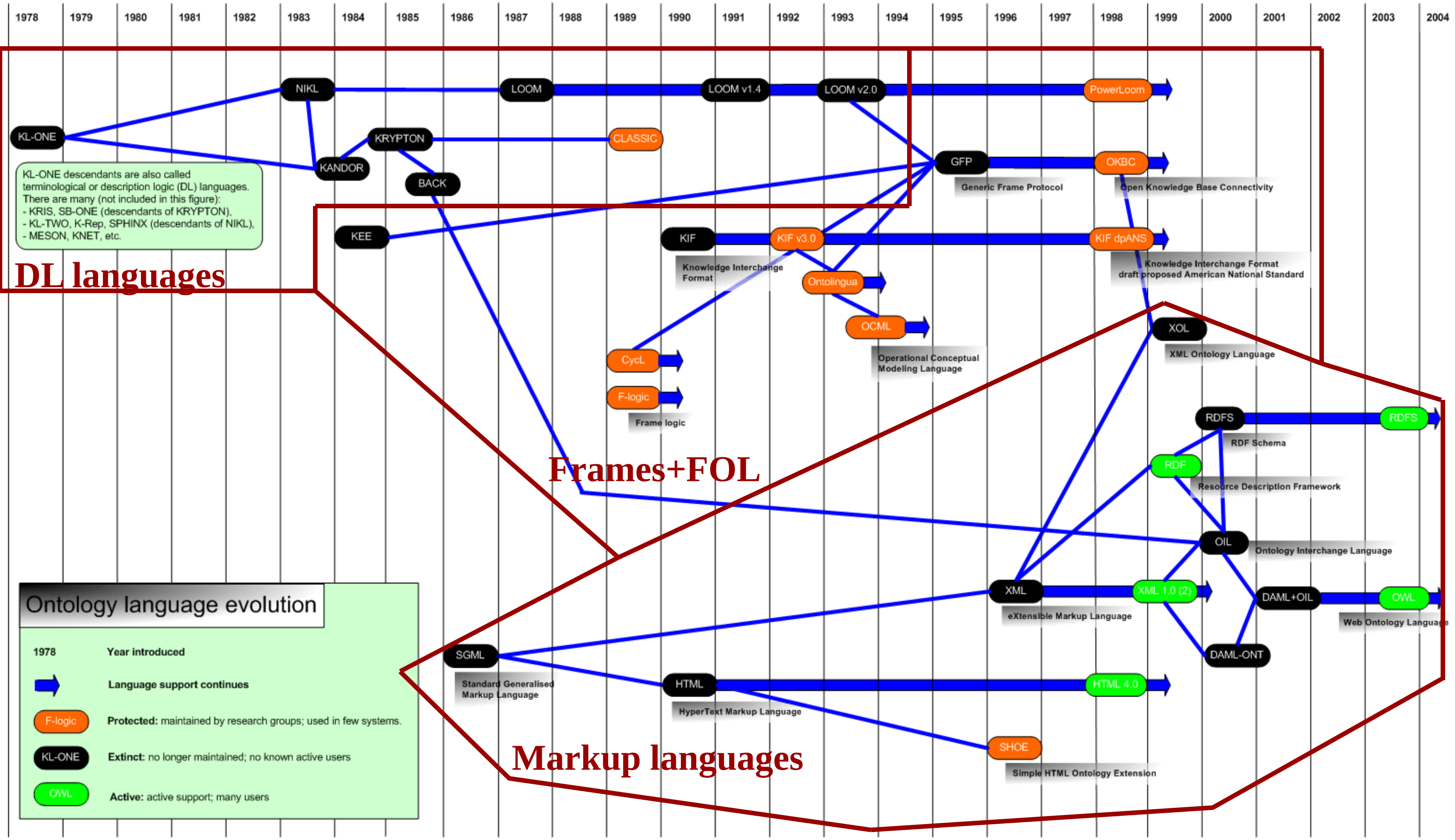
- **Asunción Gómez-Pérez and Mariano Fernández-López**
  - Most of the slides have been done jointly with them
- **Sean Bechhoffer (University of Manchester)**
  - Tableaux reasoning
  - Examples about reasoning
- **CO-ODE people (University of Manchester)**
  - http://www.co-ode.org/
  - Some RDF, RDFS and OWL descriptions
  - Use of reasoners
  - Protégé-OWL tutorial

Table of Contents

KR Formalisms



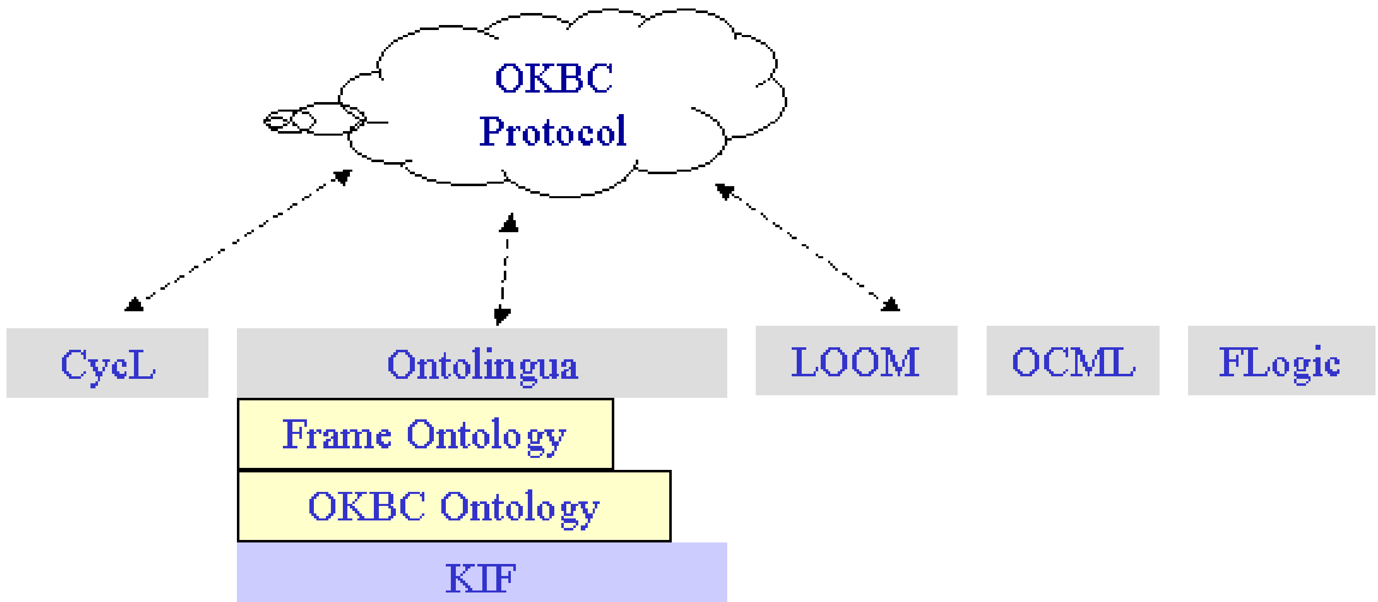
Ontology language evolution



Ontology Languages (I)

Traditional ontology languages

- **Ontolingua/KIF**
- **OKBC**
- **OCML**
- **LOOM**
- **FLogic**



Ontology Languages (II)

Ontology markup languages

- **Standards & Recommendations of W3C**
  - **XML**
  - **RDF(S)**
- **Ontology specification languages**
  - **SHOE**
  - **XOL**
  - **OIL**
  - **DAML+OIL**
  - **OWL**

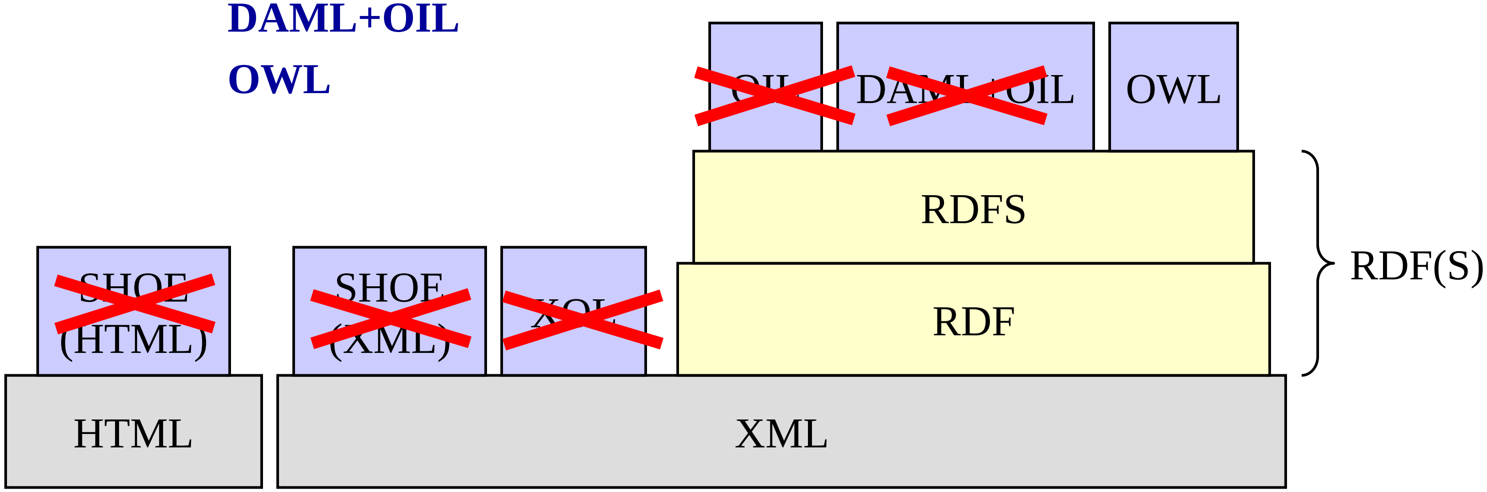
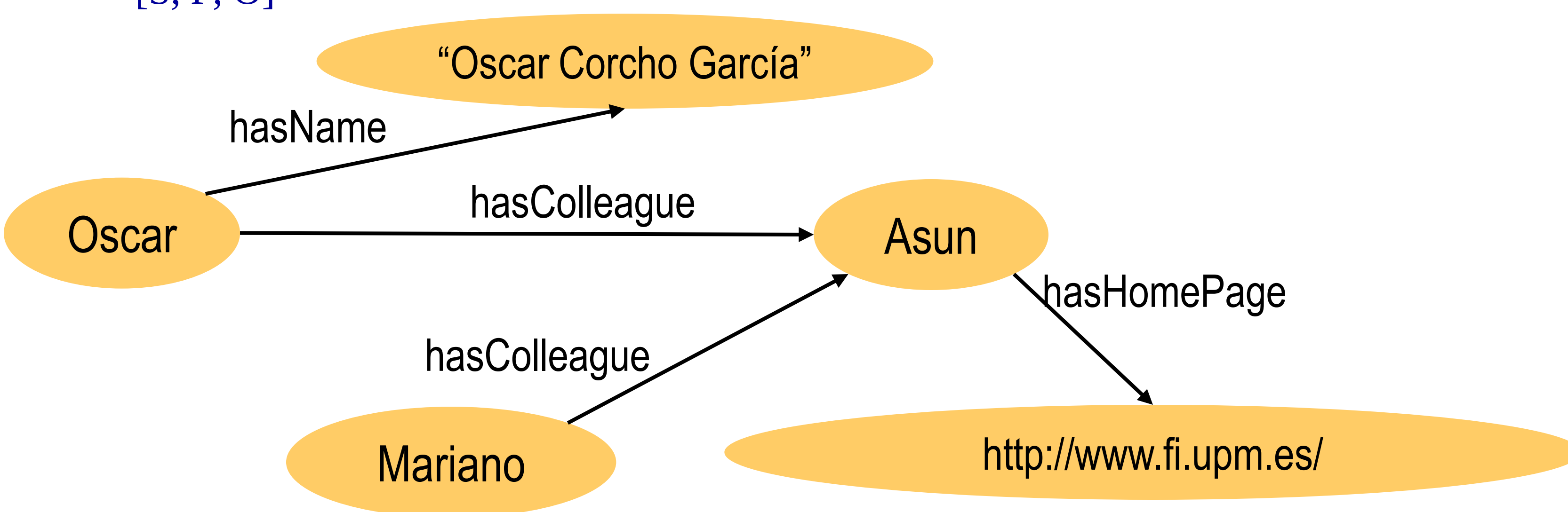


Table of Contents

RDF: Resource Description Framework

- **W3C recommendation (http://www.w3.org/RDF)**
- **RDF is graphical formalism (+ XML syntax + semantics)**
  - for representing metadata
  - for describing the semantics of information in a machine- accessible way
- **RDF is a basic ontology language**
  - Resources are described in terms of properties and property values using RDF statements.
  - Statements are represented as triples, consisting of a subject, predicate and object. [S, P, O]



RDF and URIs

- **RDF uses URIRefs (Unique Resource Identifiers References) to identify resources.**
  - A URIRef consists of a URI and an optional Fragment Identifier separated from the URI by the hash symbol #.
  - Examples
    - **http://www.co-ode.org/people#hasColleague**
    - **coode:hasColleague**

- **A set of URIRefs is known as a vocabulary**
  - E.g., the RDF Vocabulary
    - The set of URIRefs used in describing the RDF concepts: **rdf:Property**, **rdf:Resource**, **rdf:type**, etc.
  - The RDFS Vocabulary
    - The set of URIRefs used in describing the RDF Schema language: **rdfs:Class**, **rdfs:domain**, etc.
  - The 'Pizza Ontology' Vocabulary
    - **pz:hasTopping, pz:Pizza**, **pz:VegetarianPizza**, etc.

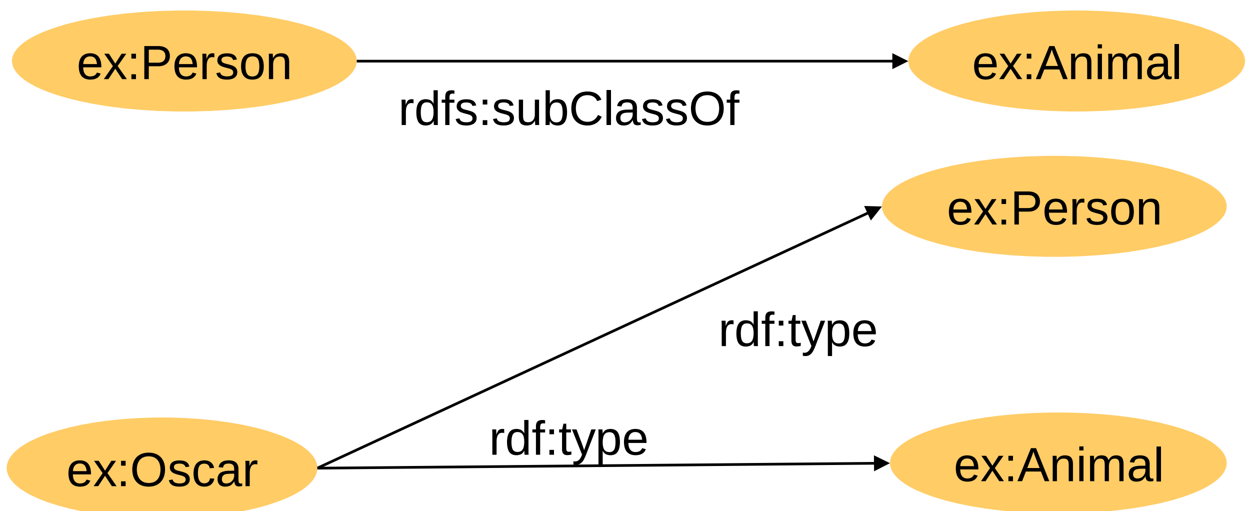
RDF Serialisation

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:coode="http://www.co-ode.org/people#"
      xml:base="http://www.co-ode.org/people">
<rdf:Description rdf:ID="mh">
         <coode:hasHomepage rdf:resource="http://www.cs.man.ac.uk/~horridgm"/>
         <coode:hasName>Matthew Horridge</coode:hasName>
</rdf:Description>
<rdf:Description rdf:ID="nd">
         <coode:hasName>Nick Drummond</coode:hasName>
         <coode:hasColleage rdf:resource="#mh"/>
</rdf:Description>
</rdf:RDF>
```

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:coode="http://www.co-ode.org/people#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xml:base="file:/Users/matthewhorridge/Desktop/Test.rdf">
  <rdf:Description rdf:about="http://www.co-ode.org/people#nd">
    <coode:hasName>Nick Drummond</coode:hasName>
    <coode:hasColleage>
      <rdf:Description rdf:about="http://www.co-ode.org/people#mh">
        <coode:hasName>Matthew Horridge</coode:hasName>
        <coode:hasHomepage rdf:resource="http://www.cs.man.ac.uk/~horridgm"/>
      </rdf:Description>
    </coode:hasColleage>
  </rdf:Description>
</rdf:RDF>
```

RDFS: RDF Schema

- **W3C Recommendation**
- **RDF Schema extends RDF to enable talking about classes of resources, and the properties to be used with them.**
  - Class definition: rdfs:Class, rdfs:subClassOf
  - Property definition: rdfs:subPropertyOf, rdfs:range, rdfs:domain
  - Other primitives: rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
- **RDF Schema provides the means to describe application specific RDF vocabularies.**
- **RDFS vocabulary adds constraints on models, e.g.:**
  - $\forall$x,y,z type(x,y) and subClassOf(y,z) $\rightarrow$ type(x,z)

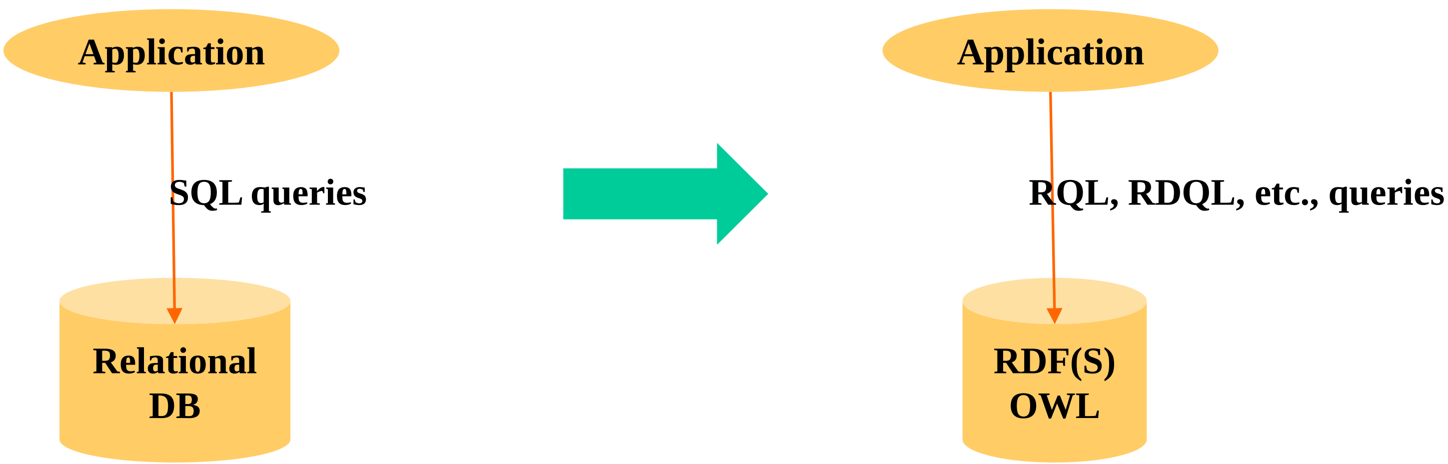


RDF(S) limitations

- **RDFS too weak to describe resources in sufficient detail**
  - No localised range and domain constraints
    - Can't say that the range of hasChild is person when applied to persons and elephant when applied to elephants
  - No existence/cardinality constraints
    - Can't say that all *instances* of person have a mother that is also a person, or that persons have exactly 2 parents
  - No transitive, inverse or symmetrical properties
    - Can't say that isPartOf is a transitive property, that hasPart is the inverse of isPartOf or that touches is symmetrical
  - …
- **Difficult to provide reasoning support**
  - No "native" reasoners for non-standard semantics
  - May be possible to reason via FO axiomatisation

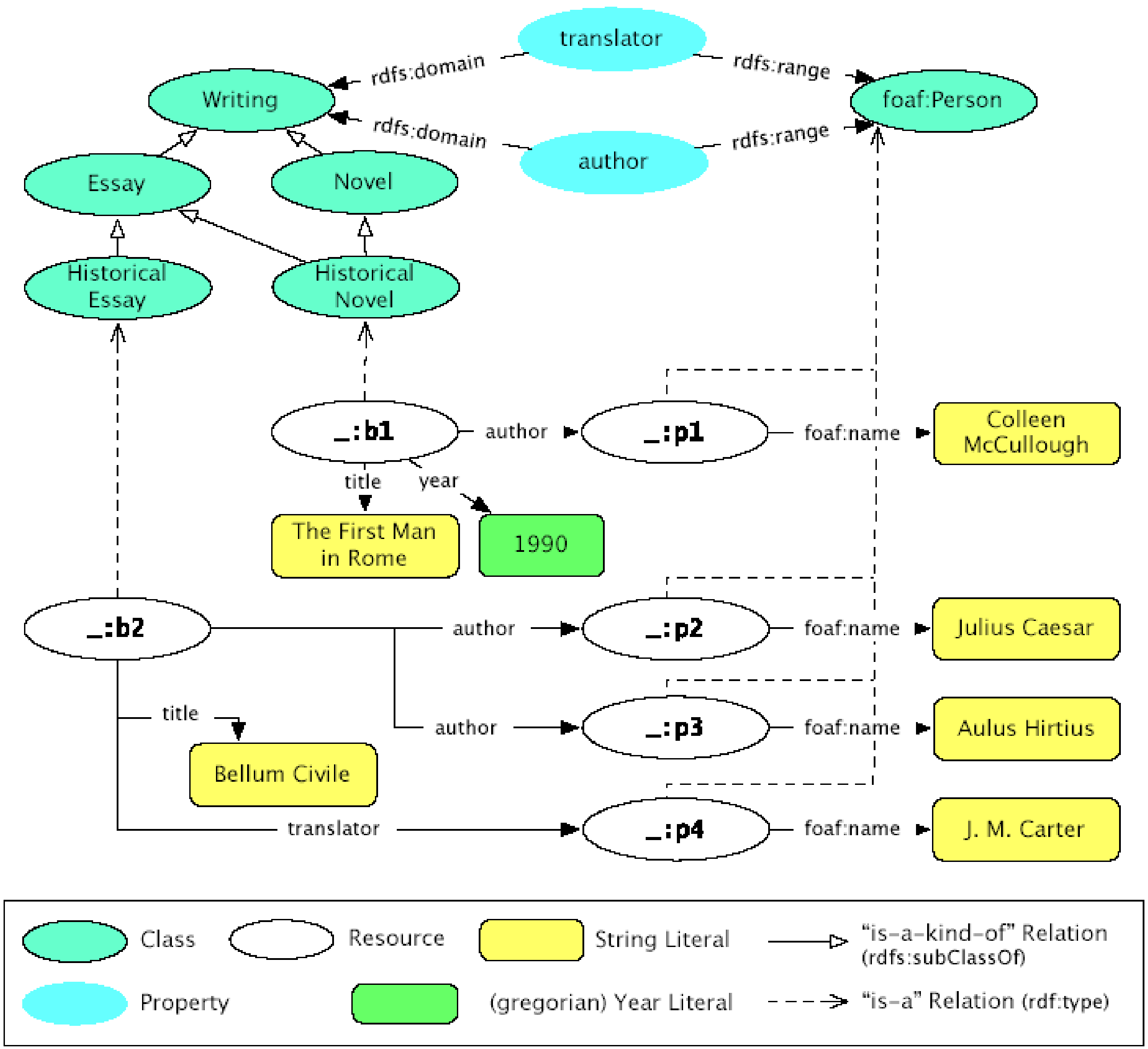
RDF(S) query languages

- **Languages developed to allow accessing datasets expressed in RDF(S) (and in some cases OWL)**



- **Supported by the most important language APIs**
  - Jena (HP labs)
  - Sesame (Aduna)
  - Boca (IBM)
  - KPOntology (iSOCO)
  - ...
- **There are some differences wrt languages like SQL, such as**
  - Combination of different sources
  - Trust management
  - Open World Assumption

Example of a RDF(S) model



Query types

- **Selection and extraction**
  - "Select all the essays, together with their authors and their authors' names".
  - "Select everything that is related to the book 'Bellum Civille'"
- **Reduction: we specify what it should not be returned**
  - "Select everything except for the ontological information and the book translators"
- **Restructuring: the original structure is changed in the final result**
  - "Invert the relationship 'author' by 'is author of'"
- **Aggregation**
  - "Return all the essays together with the mean number of authors per essay"
- **Combination and inferences**
  - "Combine the information of a book called 'La guerra civil' and whose author is Julius Caesar with the book whose identifier is 'Bellum Civille'"
  - "Select all the essays, together with its authors and author names", *including also the instances of the subclasses of Essay*.
  - "Obtain the relationship 'coauthor' among persons who have written the same book".

RDF(S) query language families

- **SquishQL Family**
  - SquishQL
  - rdfDB Query Language
  - RDQL
  - BRQL
  - TriQL
- **XPath, XSLT, Xquery**
  - XQuery for RDF
  - XsRQL
  - TreeHugger and RDFTwig
  - RDFT, Nexus Query Language
  - RDFPath, Rpath and RXPath
  - Versa
- **RQL Family**
  - RQL
  - SeRQL
  - eRQL
- **Controlled natural language**
  - Metalog
- **Other**
  - Algae
  - iTQL
  - N3QL
  - PerlRDF Query Language
  - RDEVICE Deductive Language
  - RDFQBE
  - RDFQL
  - TRIPLE
  - WQL

SPARQL

(W3C Working Draft)

RDQL

- **Supported by: Jena, RAP, PHP XML Classes, RDFStore, Rasqal RDF Query Library, Sesame, 3store, RDQLPlus**

- **Example**

  SELECT ?person
  FROM http://somewhere.org/somerdfmodelofpeople
  WHERE (?person, info:age, ?age)
  AND ?age > 24
  USING info FOR <http://example.org/peopleInfo#>

- **Features**
  - Based on triple patterns and triple conjunctions
  - It specifies the graph structure to be obtained as a result of the query

- **Limitations**
  - It does not make inferences based on the semantics of RDFS
    - Except for Jena, which supports the relations subClassOf and subPropertyOf
  - It allows mixing domain vocabulary (RDF) and knowledge representation vocabulary (RDFS) in the queries, and both types of vocabulary are treated in the same way by the query interpreter.
  - It only supports selection and extraction queries

SeRQL

- **Supported by: Sesame, SWI-Prolog**
- **Example**

  SELECT X, Y
  FROM {X}mybooks:authored{Y}books:translator{T}
  *WHERE T like "*Carter*"*
  USING NAMESPACE books = &http://example.org/books#
  USING NAMESPACE mybooks = &http://example.org/booksrdfsextension#

- **Features**
  - It supports most of the aforementioned queries (although its implementations do not necessarily support them)
  - It supports datatype reasoning (datatypes can be requested instead of actual values)
  - The domain vocabulary and the knowledge representation vocabulary are treated differently by the query interpreters.
  - It allows making queries over properties with multiple values, over multiple properties of a resource and over reifications
  - Queries can contain optional statements
- **Limitations**
  - Neither set operations nor existential or universal quantifiers can be included in the queries
  - It does not support aggregation queries nor recursive queries

SPARQL

- **Supported by: Jena, Sesame, IBM Boca, etc.**
- **Example**

```
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
SELECT ?y ?givenName
WHERE { ?y vcard:Family "Smith" . ?y vcard:Given ?givenName . }
```

- **Features**
  - It supports most of the aforementioned queries
  - It supports datatype reasoning (datatypes can be requested instead of actual values)
  - The domain vocabulary and the knowledge representation vocabulary are treated differently by the query interpreters.
  - It allows making queries over properties with multiple values, over multiple properties of a resource and over reifications
  - Queries can contain optional statements
  - It supports aggregation queries
- **Limitations**
  - Neither set operations nor existential or universal quantifiers can be included in the queries
  - It does not support recursive queries

Exercise

- **Objective**
  - **Understand the features of RDF(S) for implementing ontologies, including its limitations**
  - **Get used to the graph and XML syntax of RDF(S)**
- **Tasks**
  - **Take ontologies previously and create their graphs**
    - **First only include the vocabulary from the domain**
    - **Then include references to the RDF and RDFS vocabularies**
  - **Serialise part of the graph in the XML syntax**

dbpedia.org

- **http://dbpedia.org/**

1.2. Example queries displayed with the Leipzig query builder

- Tennis players from Moscow
- Sitcoms set in NYC
- People influenced by Friedrich Nietzsche
- Space Missions
- Soccer player with tricot number 11 from club with stadium with >40000 seats born in a country with more than 10M inhabitants

1.3. Example queries displayed with the Berlin SNORQL query explorer

- People who were born in Berlin before 1900
- German musicians with German and English descriptions
- German musicians who were born in Berlin
- French films
- Ego-shooter computer games
- Luxus cars

dbpedia.org

- **German musicians born in Berlin**

SPARQL Explorer for http://dbpedia.org/sparql

SPARQL:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <http://dbpedia.org/resource/>
PREFIX dbpedia2: <http://dbpedia.org/property/>
PREFIX dbpedia: <http://dbpedia.org/>
PREFIX skos: <http://www.w3.org/2004/02/skos/core#>
SELECT ?name ?birth ?description ?person WHERE {
     ?person dbpedia2:birthPlace <http://dbpedia.org/resource/Berlin> .
     ?person skos:subject <http://dbpedia.org/resource/Category:German_musicians> .
     ?person dbpedia2:birth ?birth .
     ?person foaf:name ?name .
     ?person rdfs:comment ?description .
     FILTER (LANG(?description) = 'en') .
}
ORDER BY ?name
```

Results: Browse Go! Reset

dbpedia.org

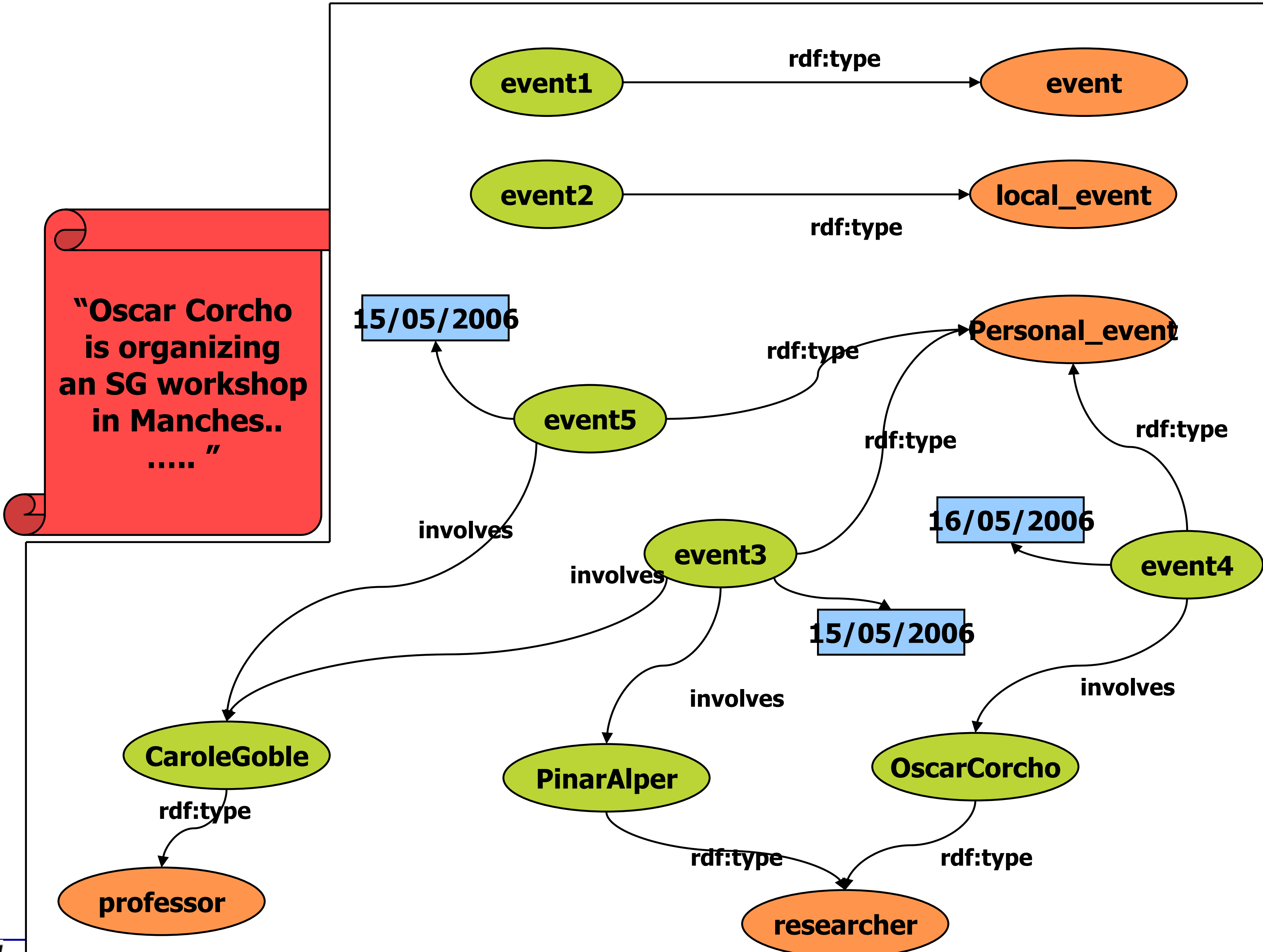
SPARQL results:

name	birth	description	person
"Moser, Edda"@de	"1938-10-27"^^xsd:date	"The German soprano Edda Moser was born on October 27, 1938 in Berlin, Germany. She is the daughter of the musicologist Hans Joachim Moser."@en	:Edda_Moser
"Möbius, Ralph Christian""@de	"1950-01-09"^^xsd:date	"Rio Reiser (January 9, 1950 - August 20, 1996), was a German rock musician and singer of the famous rock group Ton Steine Scherben. He was born Ralph Christian Möbius in Berlin and died at the age of 46 in the little German town of Fresenhagen. Rio Reiser was politically active during his whole life. In the early 70ies he participated in the squatter scene, for which he wrote the famous Rauchhaussong.""@en	:Rio_Reiser
"Straube, Karl"@de	"1873-01-06"^^xsd:date	"Montgomery Rufus Karl/Carl Siegfried Straube (January 6, 1873, Berlin - April 27, 1950, Leipzig) was a German church musician , organist, and choral conductor, famous above all for championing the abundant organ music of Max Reger."@en	:Karl_Straube
"Tricht, Käte van""@de	"1909-10-22"^^xsd:date	"Käte van Tricht (October 22, 1909–July 13, 1996), was a German organist, pianist, harpsichordist, and pedagogue.""@en	:K%C3%A4te_van_Tricht
"Urlaub, Farin"@de	"1963-10-27"^^xsd:date	"Jan Ulrich Max Vetter, better known as Farin Urlaub (like German "Fahr in Urlaub!" ("Go on holiday!"), after his love of travelling) was born on October 27, 1963 in what was then West Berlin, Germany. He is best known as the guitarist/vocalist for the German punk rock band Die Ärzte.""@en	:Farin_Urlaub
"Voormann, Klaus"@de	"1938-04-29"^^xsd:date	"Klaus Voormann (born 29 April 1938) is a German artist, musician, and record producer who was associated with the early days of The Beatles in Hamburg and later designed the cover of their album Revolver."@en	:Klaus_Voormann

Other good sources for SPARQL queries

- **http://esw.w3.org/topic/SparqlEndpoints**

- **http://sparql.semantic-web.at/snorql/**
- **http://www.w3c.es/Prensa/sparql/**



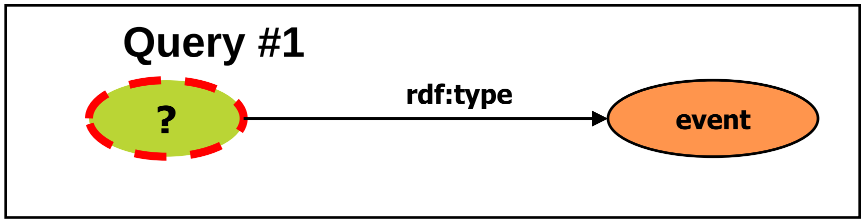
"Oscar Corcho is organizing an SG workshop in Manches.. ….. "
event1
rdf:type
event
event2
local_event
rdf:type
15/05/2006
Personal_event
rdf:type
event5
rdf:type
rdf:type
16/05/2006
involves
event3
event4
involves
15/05/2006
involves
involves
CaroleGoble
PinarAlper
OscarCorcho
rdf:type
rdf:type
rdf:type
professor
researcher

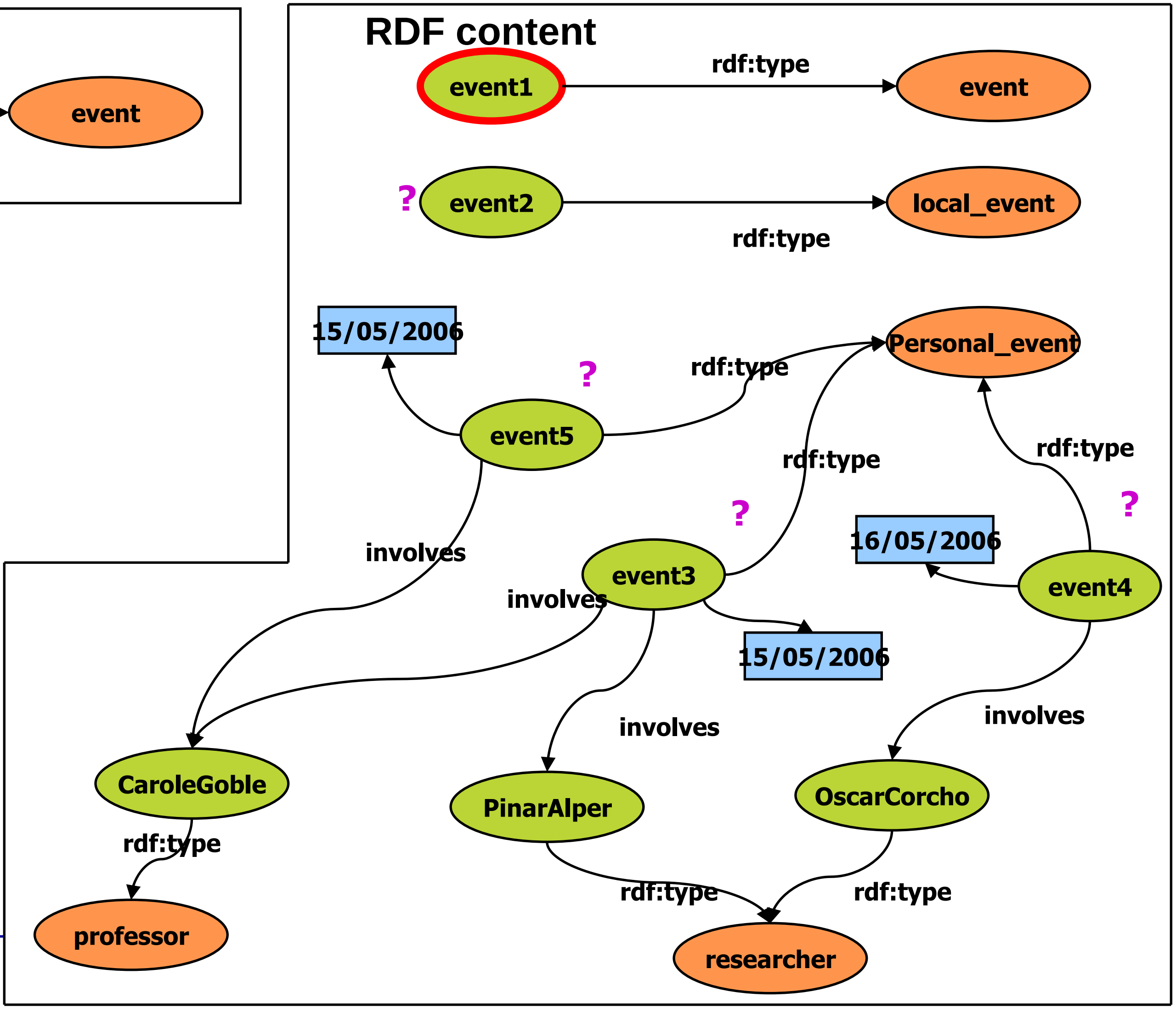
RDF querying

```
Query 1:    SELECT N
            FROM {N} rdf:type {sti:Event}
            USING NAMESPACE sti=<http://www.ontogrid.net/StickyNote#>

Query 2:    SELECT N
            FROM {N} rdf:type {sti:Event}; sti:involves {sti:OscarCorcho}
            USING NAMESPACE sti=<http://www.ontogrid.net/StickyNote#>

Query 3:    SELECT N
            FROM {N} rdf:type {sti:Event}; sti:involves {M} rdf:type {sti:Professor}
            USING NAMESPACE sti=<http://www.ontogrid.net/StickyNote#>
```

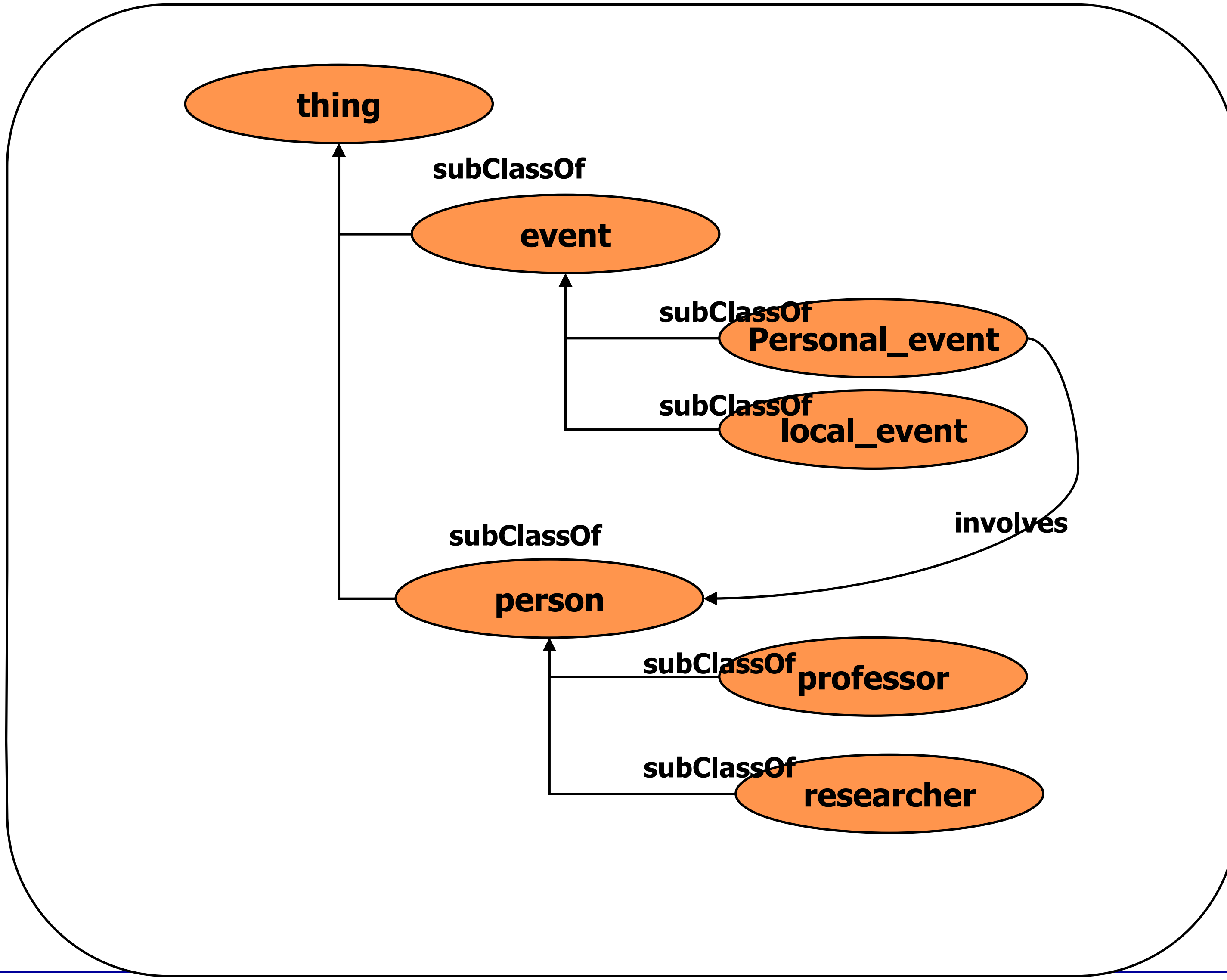
RDF Querying: why do we get “bad” results?

- Relationships between terms are missing





RDF(S) querying



RDF(S) querying: inferences

Query 1: SELECT N
FROM {N} rdf:type {sti:Event}
USING NAMESPACE sti=<http://www.ontogrid.net/StickyNote#>

Query 2: SELECT N
FROM {N} rdf:type {sti:Event}; sti:involves {sti:OscarCorcho}
USING NAMESPACE sti=<http://www.ontogrid.net/StickyNote#>

Query 3: SELECT N
FROM {N} rdf:type {sti:Event}; sti:involves {M} rdf:type {sti:Professor}
USING NAMESPACE sti=<http://www.ontogrid.net/StickyNote#>

RDF Semantics says:
(http://www.w3.org/TR/rdf-mt/)

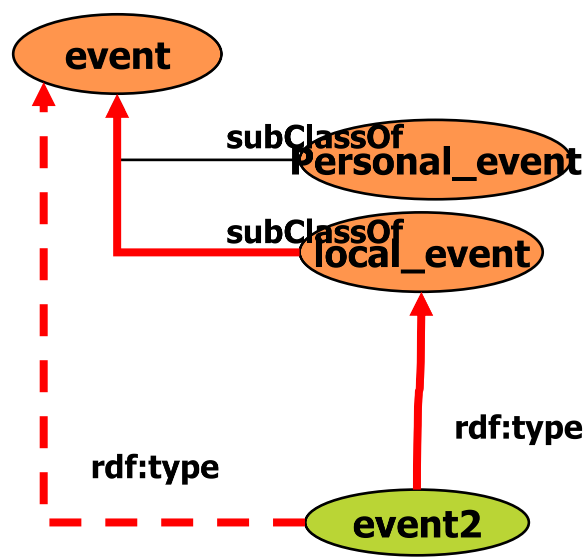


Table of Contents

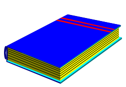
OWL

Web Ontology Language

Built on top of RDF(S) and renaming DAML+OIL primitives

3 layers:
-OWL Lite: a small subset, easier for frame-based tools to transition to, easier reasoning
-OWL DL: description logic, decidable reasoning
-OWL Full: RDF extension, allows metaclasses

Several syntaxes:
-Abstract syntax: easier to read and write manually, closely corresponds to DL
-RDF/XML: OWL can be parsed as an RDF document, more verbose

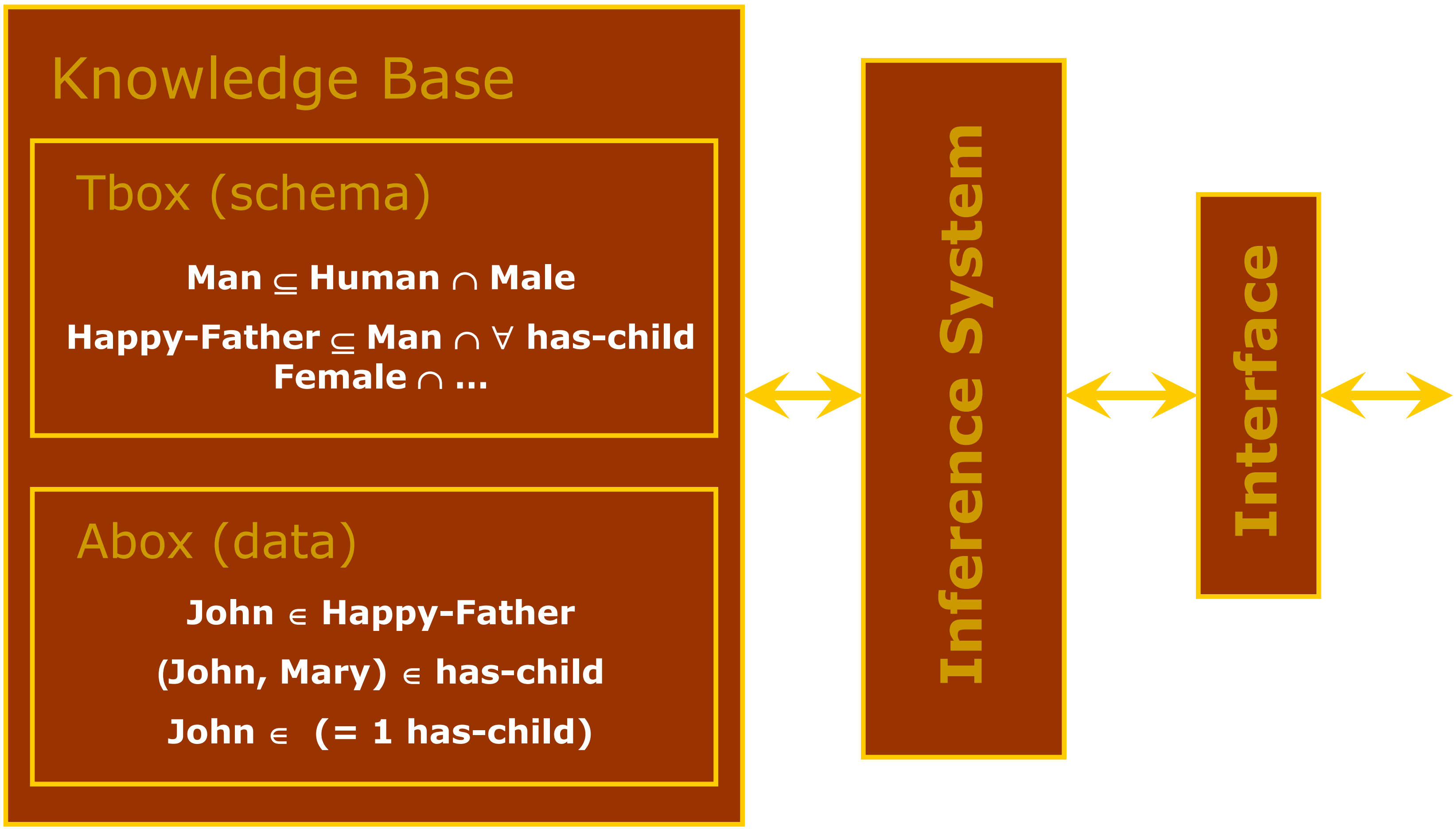


Dean M, Schreiber G. The OWL Web Ontology Language. W3C Recommendation. February 2004.

OWL and Description Logic

- **A family of logic based Knowledge Representation formalisms**
  - Descendants of semantic networks and KL-ONE
  - Describe domain in terms of concepts (classes), roles (relationships) and individuals
    - Specific languages characterised by the constructors and axioms used to assert knowledge about classes, roles and individuals.
    - Example: ALC (the least expressive language in DL that is propositionally closed)
      - Constructors: boolean (and, or, not)
      - Role restrictions
- **Distinguished by:**
  - Formal semantics (typically model theoretic)
    - Decidable fragments of FOL
    - Closely related to Propositional Modal & Dynamic Logics
  - Provision of inference services
    - Sound and complete decision procedures for key problems
    - Implemented systems (highly optimised)

DL Architecture



DL constructors

<table>
<tr><th>Construct</th><th>Syntax</th><th colspan="4">Language</th></tr>
<tr><td>Concept</td><td>A</td><td rowspan="4">FL_0</td><td rowspan="7">FL^-</td><td rowspan="8">AL</td><td rowspan="11">S[14]</td></tr>
<tr><td>Role name</td><td>R</td></tr>
<tr><td>Intersection</td><td>$C \cap D$</td></tr>
<tr><td>Value restriction</td><td>$\forall R.C$</td></tr>
<tr><td>Limited existential quantification</td><td>$\exists R$</td><td rowspan="3"></td></tr>
<tr><td>Top or Universal</td><td>$\top$</td></tr>
<tr><td>Bottom</td><td>$\bot$</td></tr>
<tr><td>Atomic negation</td><td>$\neg A$</td><td colspan="2"></td></tr>
<tr><td>Negation[15]</td><td>$\neg C$</td><td colspan="3">C</td></tr>
<tr><td>Union</td><td>$C \cup D$</td><td colspan="3">U</td></tr>
<tr><td>Existential restriction</td><td>$\exists R.C$</td><td colspan="3">E</td></tr>
<tr><td>Number restrictions</td><td>$(\geq n\ R)\ \ (\leq n\ R)$</td><td colspan="4">N</td></tr>
<tr><td>Nominals</td><td>$\{a_1 \ldots a_n\}$</td><td colspan="4">O</td></tr>
<tr><td>Role hierarchy</td><td>$R \subseteq S$</td><td colspan="4">H</td></tr>
<tr><td>Inverse role</td><td>R^-</td><td colspan="4">I</td></tr>
<tr><td>Qualified number restriction</td><td>$(\geq n\ R.C)\ \ (\leq n\ R.C)$</td><td colspan="4">Q</td></tr>
</table>

OWL is SHOIN(D+)

→ ≥3 hasChild, ≤1 hasMother

→ {Colombia, Argentina, México, ...} → MercoSur countries

→ $hasChild^-$ (hasParent)

→ ≤2 hasChild.Female, ≥1 hasParent.Male

Other:

Concrete datatypes: hasAge.(<21)

Transitive roles: hasChild* (descendant)

Role composition: hasParent o hasBrother (uncle)

[12] Names previously used for Description Logics were: terminological knowledge representation languages, concept languages, term subsumption languages, and KL-ONE-based knowledge representation languages.

[13] In this table, we use A to refer to atomic concepts (concepts that are the basis for building other concepts), C and D to any concept definition, R to atomic roles and S to role definitions. FL is used for structural DL languages and AL for attributive languages (Baader et al., 2003).

[14] S is the name used for the language ALC_{R+}, which is composed of ALC plus transitive roles.

[15] ALC and ALCUE are equivalent languages, since union (U) and existential restriction (E) can be represented using negation (C).

OWL as a Description Logic language. Class constructors

Constructor	DL Syntax	Example	Modal Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1 \land \ldots \land C_n$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1 \lor \ldots \lor C_n$
complementOf	$\neg C$	$\neg$Male	$\neg C$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x_1 \lor \ldots \lor x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$[P]C$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\langle P \rangle C$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$[P]_{n+1}$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\langle P \rangle_n$

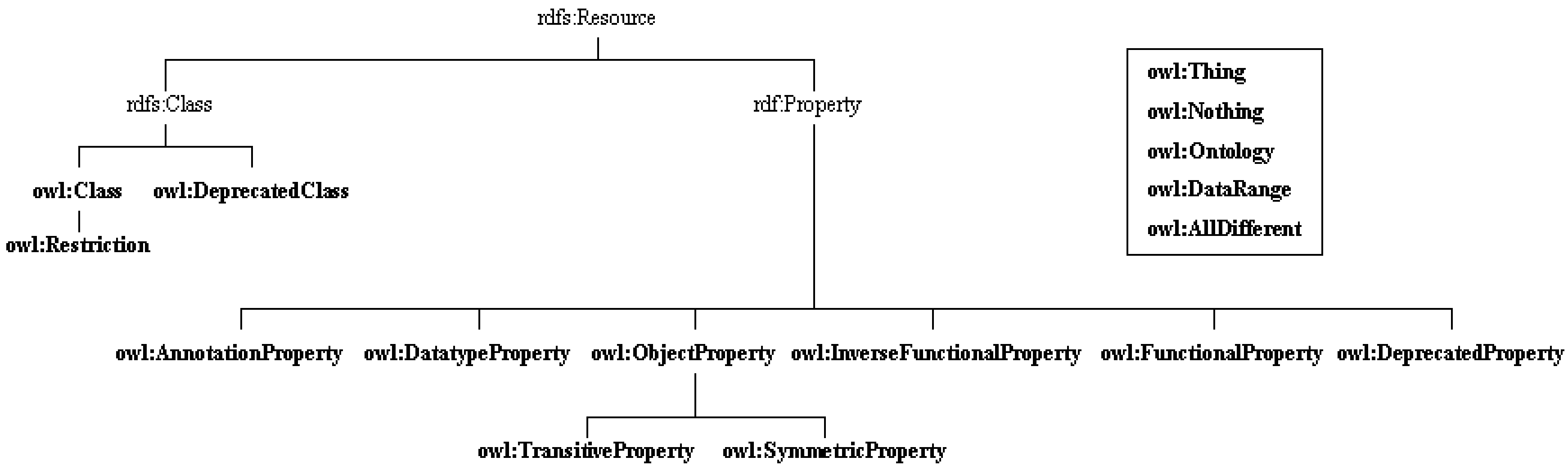
- **XML Schema datatypes are treated as classes**
  - $\forall$hasAge.nonNegativeInteger
- **Nesting of constructors can be arbitrarily complex**
  - Person $\land$ $\forall$hasChild.(Doctor $\lor$ $\exists$hasChild.Doctor)
- **Lots of redundancy, e.g., use negations to transform** and **to** or **and** exists **to** forall

OWL Axioms

Axiom	DL Syntax	Example
subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
disjointWith	$C_1 \sqsubseteq \neg C_2$	Male $\sqsubseteq$ $\neg$Female
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$	{President_Bush} $\equiv$ {G_W_Bush}
differentFrom	$\{x_1\} \sqsubseteq \neg\{x_2\}$	{john} $\sqsubseteq$ $\neg${peter}
subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
inverseOf	$P_1 \equiv P_2^-$	hasChild $\equiv$ hasParent$^-$
transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor
functionalProperty	$\top \sqsubseteq \leqslant 1 P$	$\top \sqsubseteq \leqslant$1hasMother
inverseFunctionalProperty	$\top \sqsubseteq \leqslant 1 P^-$	$\top \sqsubseteq \leqslant$1hasSSN$^-$

- **Axioms (mostly) reducible to inclusion (v)**
  - C $\equiv$ D iff both C $\subseteq$ D and D $\subseteq$ C
  - C disjoint D iff C $\subseteq$ $\neg$D

Class taxonomy of the OWL KR ontology



Property list of the OWL KR ontology

Property name	domain	range
owl:intersectionOf	owl:Class	rdf:List
owl:unionOf	owl:Class	rdf:List
owl:complementOf	owl:Class	owl:Class
owl:oneOf	owl:Class	rdf:List
owl:onProperty	owl:Restriction	rdf:Property
owl:allValuesFrom	owl:Restriction	rdfs:Class
owl:hasValue	owl:Restriction	*not specified*
owl:someValuesFrom	owl:Restriction	rdfs:Class
owl:minCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite: {0,1} OWL DL/Full: {0,..,N}
owl:maxCardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite: {0,1} OWL DL/Full: {0,..,N}
owl:cardinality	owl:Restriction	xsd:nonNegativeInteger OWL Lite: {0,1} OWL DL/Full: {0,..,N}
owl:inverseOf	owl:ObjectProperty	owl:ObjectProperty
owl:sameAs	owl:Thing	owl:Thing
owl:equivalentClass	owl:Class	owl:Class
owl:equivalentProperty	rdf:Property	rdf:Property
owl:sameIndividualAs	owl:Thing	owl:Thing
owl:differentFrom	owl:Thing	owl:Thing
owl:disjointWith	owl:Class	owl:Class
owl:distinctMembers	owl:AllDifferent	rdf:List
owl:versionInfo	*not specified*	*not specified*
owl:priorVersion	owl:Ontology	owl:Ontology
owl:incompatibleWith	owl:Ontology	owl:Ontology
owl:backwardCompatibleWith	owl:Ontology	owl:Ontology
owl:imports	owl:Ontology	owl:Ontology

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom

Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*, restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*, restricted to {0,1}),
owl:cardinality (*daml:cardinality*, restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

R

$R \subseteq S$

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

$C \subseteq D$

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifier
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

OWL Lite
owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

$\forall R.C$

$\exists R.C$

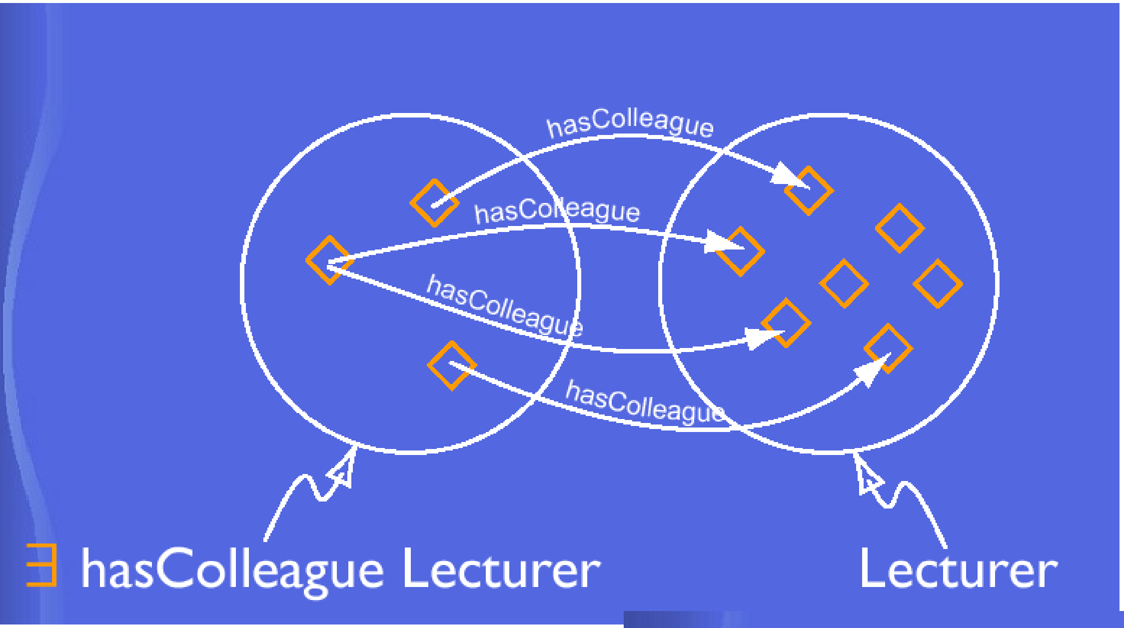
$\leq nR$

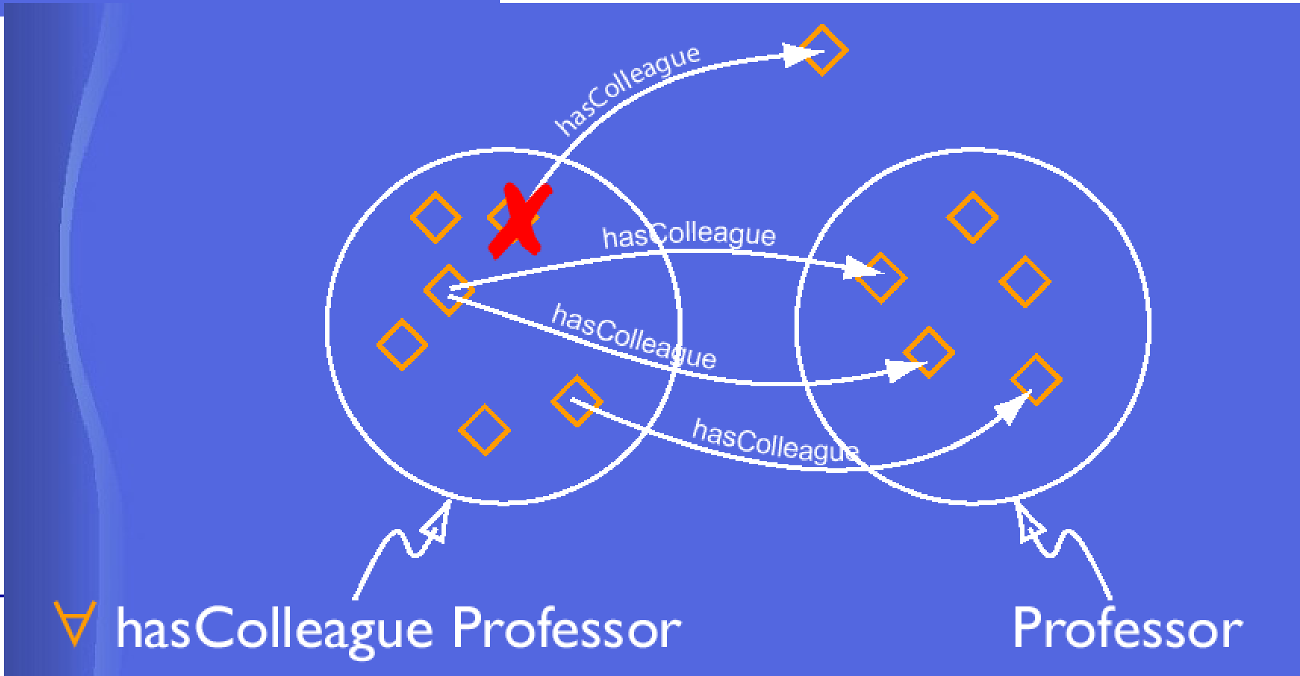
$\geq nR$

$= nR$

$C \cap D$

Existential and Universal Restrictions





OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValue

Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*; restricted to {0,1}),
owl:maxCardinality (*daml:maxCardinality*; restricted to {0,1}),
owl:cardinality (*daml:cardinality*; restricted to {0,1})

owl:intersectionOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*)
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*), $\rightarrow$ $R^{(+)}$
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

owl:Thing (*daml:Thing*) $\rightarrow$ $\top$
owl:Nothing (*daml:Nothing*) $\rightarrow$ $\bot$

owl:inverseOf (*daml:inverseOf*), $\rightarrow$ R^{-1}
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions), $\rightarrow$ $C \equiv D$
owl:equivalentProperty (*daml:samePropertyAs*), $\rightarrow$ $R \equiv S$
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

$\rightarrow$ *Abox*

OWL DL

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

OWL Lite

owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

Class expressions allowed in: rdfs:domain, rdfs:range, rdfs:subClassOf
owl:intersectionOf, owl:equivalentClass, owl:allValuesFrom, owl:someValuesFrom
Values are not restricted (0..N) in: owl:minCardinality, owl:maxCardinality, owl:cardinality

owl:DataRange, rdf:List, rdf:first, rdf:rest, rdf:nil

owl:hasValue (*daml:hasValue*)
owl:oneOf (*daml:oneOf*)
owl:unionOf (*daml:unionOf*), owl:complementOf (*daml:complementOf*)
owl:disjointWith (*daml:disjointWith*)

$\exists R.\{x\}$

$\{x\}$

$\neg C$

$C \cup D$

$C \cap D \subseteq \bot$

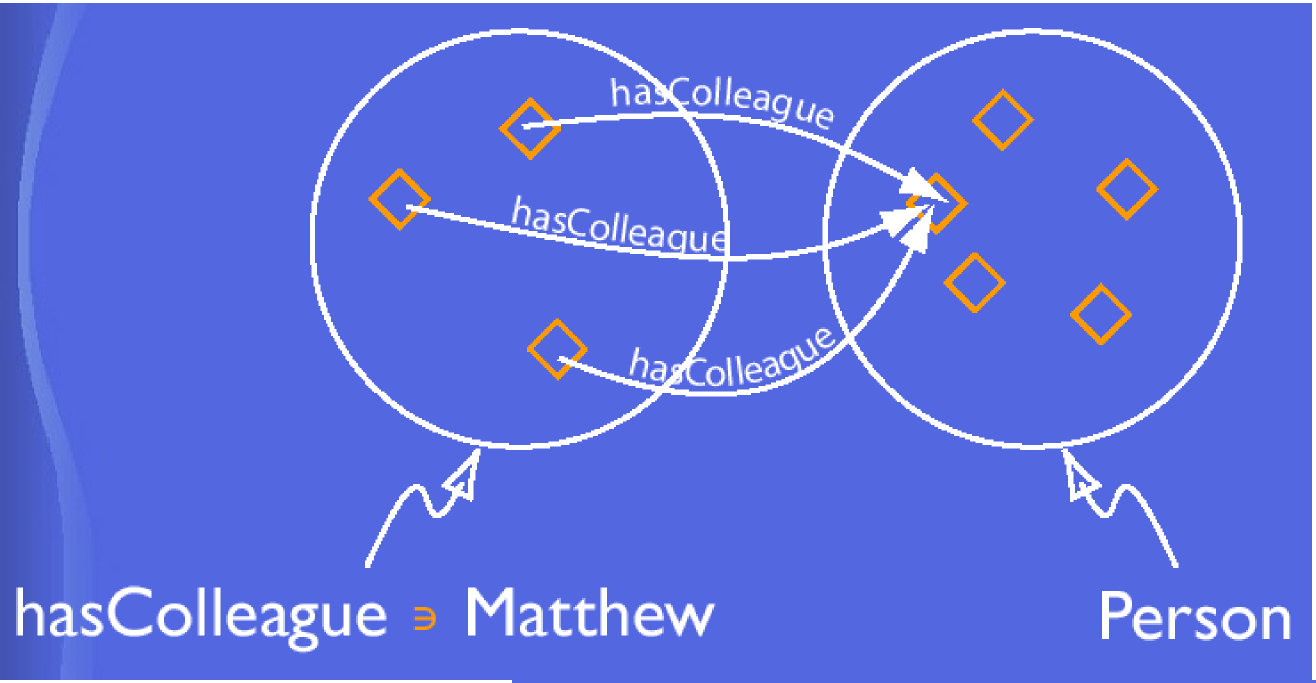
owl:Nothing (*daml:Nothing*)

owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent, owl:distinctMembers

RDF(S)

rdf:Property
rdfs:subPropertyOf
rdfs:domain
rdfs:range (only with class identifiers and named datatypes)
rdfs:comment, rdfs:label, rdfs:seeAlso, rdfs:isDefinedBy
rdfs:subClassOf (only with class identifiers and property restrictions)

hasValue and oneOf



To specify an enumerated class, the individuals that are members of the class are listed inside curly brackets {...}

{Spain Germany France Italy}

OWL Example

Develop a sample ontology in the domain of people, pets, vehicles, and newspapers

- **Practice with DL syntax, OWL abstract syntax and OWL RDF/XML syntax**
- **Understand the basic primitives of OWL Lite and OWL DL**
- **Understand the basic reasoning mechanisms of OWL DL (tomorrow)**
  - **Subsumption**
    - **Automatic classification: an ontology built collaboratively**
    - **Instance classification**
    - **Detecting redundancy**
  - **Consistency checking: unsatisfiable restrictions in a Tbox (are the classes coherent?)**

Some basic DL modelling guidelines

- **X must be Y, X is an Y that...** → $X \subseteq Y$
- **X is exactly Y, X is the Y that...** → $X \equiv Y$
- **X is not Y (*not the same as X is whatever it is not Y*)** → $X \subseteq \neg Y$
- **X and Y are disjoint** → $X \cap Y \subseteq \bot$
- **X is Y or Z** → $X \subseteq Y \cup Z$
- **X is Y for which property P has only instances of Z as values** → $X \subseteq Y \cap (\forall P.Z)$
- **X is Y for which property P has at least an instance of Z as a value** → $X \subseteq Y \cap (\exists P.Z)$
- **X is Y for which property P has at most 2 values** → $X \subseteq Y \cap (\leq 2.P)$
- **Individual X is a Y** → $X \in Y$

Chunk 1. Formalize in DL, and then in OWL DL

1. **Concept definitions**:

   Grass and trees must be plants. Leaves are parts of a tree but there are other parts of a tree that are not leaves. A dog must eat bones, at least. A sheep is an animal that must only eat grass. A giraffe is an animal that must only eat leaves. A mad cow is a cow that eats brains that can be part of a sheep.

2. **Restrictions**:

   Animals or part of animals are disjoint with plants or parts of plants.

3. **Properties**:

   Eats is applied to animals. Its inverse is eaten_by.

4. **Individuals**:

   Tom.
   Flossie is a cow.
   Rex is a dog and is a pet of Mick.
   Fido is a dog.
   Tibbs is a cat.

Chunk 2. Formalize in DL, and then in OWL DL

1. **Concept definitions**:

   Bicycles, buses, cars, lorries, trucks and vans are vehicles. There are several types of companies: bus companies and haulage companies.

   An elderly person must be adult. A kid is (exactly) a person who is young. A man is a person who is male and is adult. A woman is a person who is female and is adult. A grown up is a person who is an adult. And old lady is a person who is elderly and female. Old ladies must have some animal as pets and all their pets are cats.

2. **Restrictions**:

   Youngs are not adults, and adults are not youngs.

3. **Properties**:

   Has mother and has father are subproperties of has parent.

4. **Individuals**:

   Kevin is a person.
   Fred is a person who has a pet called Tibbs.
   Joe is a person who has at most one pet. He has a pet called Fido.
   Minnie is a female, elderly, who has a pet called Tom.

Chunk 3. Formalize in DL, and then in OWL DL

1. **Concept definitions**:

   A magazine is a publication. Broadsheets and tabloids are newspapers. A quality broadsheet is a type of broadsheet. A red top is a type of tabloid. A newspaper is a publication that must be either a broadsheet or a tabloid.

   White van mans must read only tabloids.

2. **Restrictions**:

   Tabloids are not broadsheets, and broadsheets are not tabloids.

3. **Properties**:

   The only things that can be read are publications.

4. **Individuals**:

   Daily Mirror

   The Guardian and The Times are broadsheets

   The Sun is a tabloid

Chunk 4. Formalize in DL, and then in OWL DL

1. **Concept definitions**:

   A pet is a pet of something. An animal must eat something. A vegetarian is an animal that does not eat animals nor parts of animals. Ducks, cats and tigers are animals. An animal lover is a person who has at least three pets. A pet owner is a person who has animal pets. A cat liker is a person who likes cats. A cat owner is a person who has cat pets. A dog liker is a person who likes dogs. A dog owner is a person who has dog pets.

2. **Restrictions**:

   Dogs are not cats, and cats are not dogs.

3. **Properties**:

   Has pet is defined between persons and animals. Its inverse is is_pet_of.

4. **Individuals**:

   Dewey, Huey, and Louie are ducks.
   Fluffy is a tiger.
   Walt is a person who has pets called Huey, Louie and Dewey.

Chunk 5. Formalize in DL, and then in OWL DL

1. **Concept definitions**

   A driver must be adult. A driver is a person who drives vehicles. A lorry driver is a person who drives lorries. A haulage worker is who works for a haulage company or for part of a haulage company. A haulage truck driver is a person who drives trucks ans works for part of a haulage company. A van driver is a person who drives vans. A bus driver is a person who drives buses. A white van man is a man who drives white things and vans.

2. **Restrictions**:

   --

3. **Properties**:

   The service number is an integer property with no restricted domain

4. **Individuals**:

   Q123ABC is a van and a white thing.
   The42 is a bus whose service number is 42.
   Mick is a male who read Daily Mirror and drives Q123ABC.

Chunk 1. Formalisation in DL

$$grass \subseteq plant$$

$$tree \subseteq plant$$

$$leaf \subseteq \exists partOf.tree$$

$$dog \subseteq \exists eats.bone$$

$$sheep \subseteq animal \cap \forall eats.grass$$

$$giraffe \subseteq animal \cap \forall eats.leaf$$

$$madCow \equiv cow \cap \exists eats.(brain \cap \exists partOf.sheep)$$

$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \perp$$

Chunk 2. Formalisation in DL

$$bicycle \subseteq vehicle; bus \subseteq vehicle; car \subseteq vehicle; lorry \subseteq vehicle; truck \subseteq vehicle$$

$$busCompany \subseteq company; haulageCompany \subseteq company$$

$$elderly \subseteq person \cap adult$$

$$kid \equiv person \cap young$$

$$man \equiv person \cap male \cap adult$$

$$woman \equiv person \cap female \cap adult$$

$$grownUp \equiv person \cap adult$$

$$oldLady \equiv person \cap female \cap elderly$$

$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$

$$young \cap adult \subseteq \perp$$

$$hasMother \subseteq hasParent$$

$$hasFather \subseteq hasParent$$

Chunk 3. Formalisation in DL

$$magazine \subseteq publication$$

$$broadsheet \subseteq newspaper$$

$$tabloid \subseteq newspaper$$

$$qualityBroadsheet \subseteq broadsheet$$

$$redTop \subseteq tabloid$$

$$newspaper \subseteq publication \cap (broadsheet \cup tabloid)$$

$$whiteVanMan \subseteq \forall reads.tabloid$$

$$tabloid \cap broadsheet \subseteq \perp$$

Chunk 4. Formalisation in DL

$$pet \equiv \exists isPetOf.\mathrm{T}$$

$$animal \subseteq \exists eats.\mathrm{T}$$

$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap \forall eats.\neg(\exists partOf.animal)$$

$$duck \subseteq animal; cat \subseteq animal; tiger \subseteq animal$$

$$animalLover \equiv person \cap (\geq 3 hasPet)$$

$$petOwner \equiv person \cap \exists hasPet.animal$$

$$catLike \equiv person \cap \exists likes.cat; catOwner \equiv person \cap \exists hasPet.cat$$

$$dogLike \equiv person \cap \exists likes.dog; dogOwner \equiv person \cap \exists hasPet.dog$$

$$dog \cap cat \subseteq \perp$$

Chunk 5. Formalisation in DL

$$driver \subseteq adult$$

$$driver \equiv person \cap \exists drives.vehicle$$

$$lorryDriver \equiv person \cap \exists drives.lorry$$

$$haulageWorke \equiv \exists worksFor.(haulageCompany \cup \exists partOf.haulageCompany)$$

$$haulageTruckDriver \equiv person \cap \exists drives.truck \cap \\ \exists worksFor.(\exists partOf.haulageCompany)$$

$$vanDriver \equiv person \cap \exists drives.van$$

$$busDriver \equiv person \cap \exists drives.bus$$

$$whiteVanMan \equiv man \cap \exists drives.(whiteThing \cap van)$$

OWL Example

Basic Inference Tasks

- **Subsumption – check knowledge is correct (captures intuitions)**
  - Does C subsume D w.r.t. ontology O? (in *every* model I of O, $C^I \subseteq D^I$)
- **Equivalence – check knowledge is minimally redundant (no unintended synonyms)**
  - Is C equivalent to D w.r.t. O? (in *every* model I of O, $C^I = D^I$)
- **Consistency – check knowledge is meaningful (classes can have instances)**
  - Is C satisfiable w.r.t. O? (there exists *some* model I of O s.t. $C^I \neq \emptyset$)
- **Instantiation and querying**
  - Is x an instance of C w.r.t. O? (in *every* model I of O, $x^I \in C^I$)
  - Is (x,y) an instance of R w.r.t. O? (in *every* model I of O, $(x^I,y^I) \in R^I$)
- **All reducible to KB satisfiability or concept satisfiability w.r.t. a KB**
- **Can be decided using highly optimised tableaux reasoners**

Tableaux Algorithms

- **Try to prove satisfiability of a knowledge base**
- **How do they work**
  - They try to build a model of input concept C
    - Tree model property
      - If there is a model, then there is a tree shaped model
    - If no tree model can be found, then input concept unsatisfiable
  - Decompose C syntactically
    - Work on concepts in negation normal form (De Morgan's laws)
    - Use of tableaux expansion rules
    - If non-deterministic rules are applied, then there is search
  - Stop (and backtrack) if clash
    - E.g. A(x), $\neg$A(x)
  - Blocking (cycle check) ensures termination for more expressive logics

- **The algorithm finishes when no more rules can be applied or a conflict is detected**

Tableaux rules for ALC and for transitive roles

$x \bullet \{C_1 \sqcap C_2, \dots\}$	$\rightarrow_{\sqcap}$	$x \bullet \{C_1 \sqcap C_2, C_1, C_2, \dots\}$
$x \bullet \{C_1 \sqcup C_2, \dots\}$	$\rightarrow_{\sqcup}$	$x \bullet \{C_1 \sqcup C_2, C, \dots\}$ for $C \in \{C_1, C_2\}$
$x \bullet \{\exists R.C, \dots\}$	$\rightarrow_{\exists}$	$x \bullet \{\exists R.C, \dots\} \xrightarrow{R} y \bullet \{C\}$
$x \bullet \{\forall R.C, \dots\} \xrightarrow{R} y \bullet \{\dots\}$	$\rightarrow_{\forall}$	$x \bullet \{\forall R.C, \dots\} \xrightarrow{R} y \bullet \{C, \dots\}$
$x \bullet \{\forall R.C, \dots\} \xrightarrow{R} y \bullet \{\dots\}$	$\rightarrow_{\forall_+}$	$x \bullet \{\forall R.C, \dots\} \xrightarrow{R} y \bullet \{\forall R.C, \dots\}$

Tableaux examples and exercises

- Example
  - $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)$
- **Exercise 1**
  - $\exists R.(\exists R.D) \wedge \exists S.\neg D \wedge \forall S.(\exists R.D)$
- **Exercise 2**
  - $\exists R.(C \vee D) \wedge \forall R.\neg C \wedge \neg\exists R.D$

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

$$\mathcal{L}(w) = \{\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$$

w

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

$$\mathcal{L}(w) = \{\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$$

w

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

$$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$$

$$w$$

Tableaux Algorithm — Example

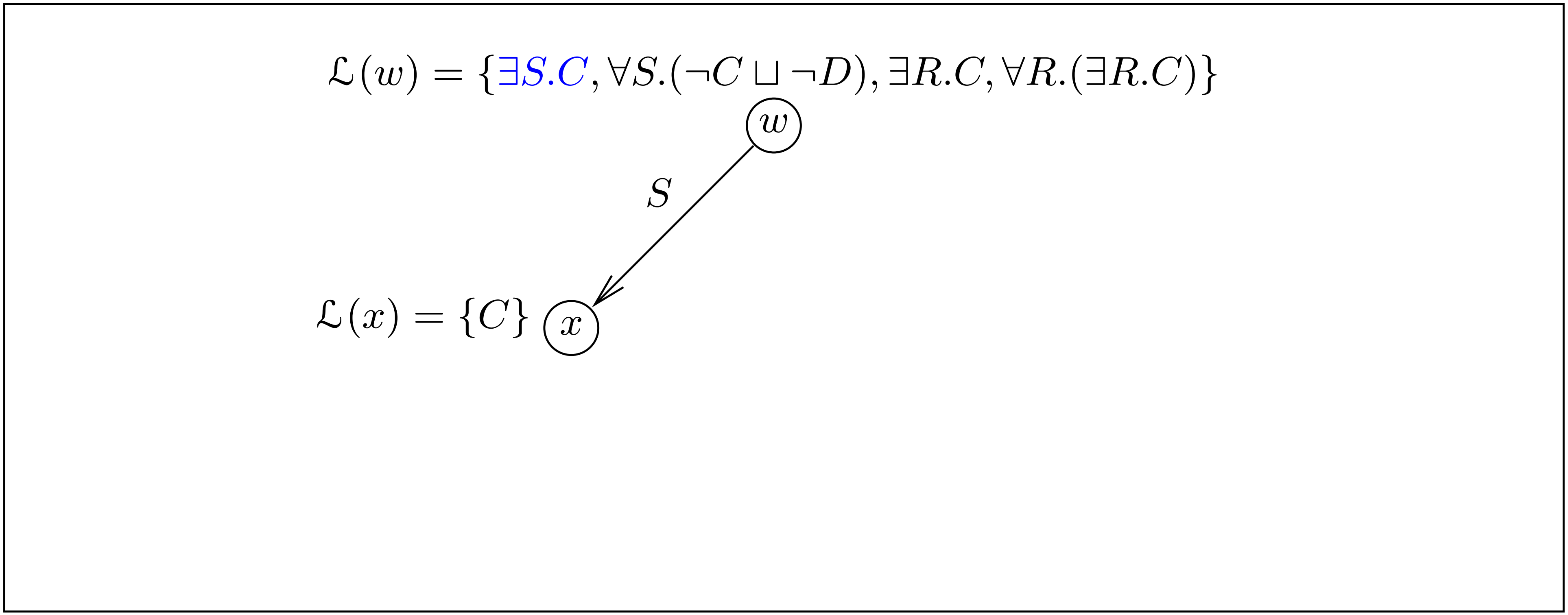
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

$$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$$

$$w$$

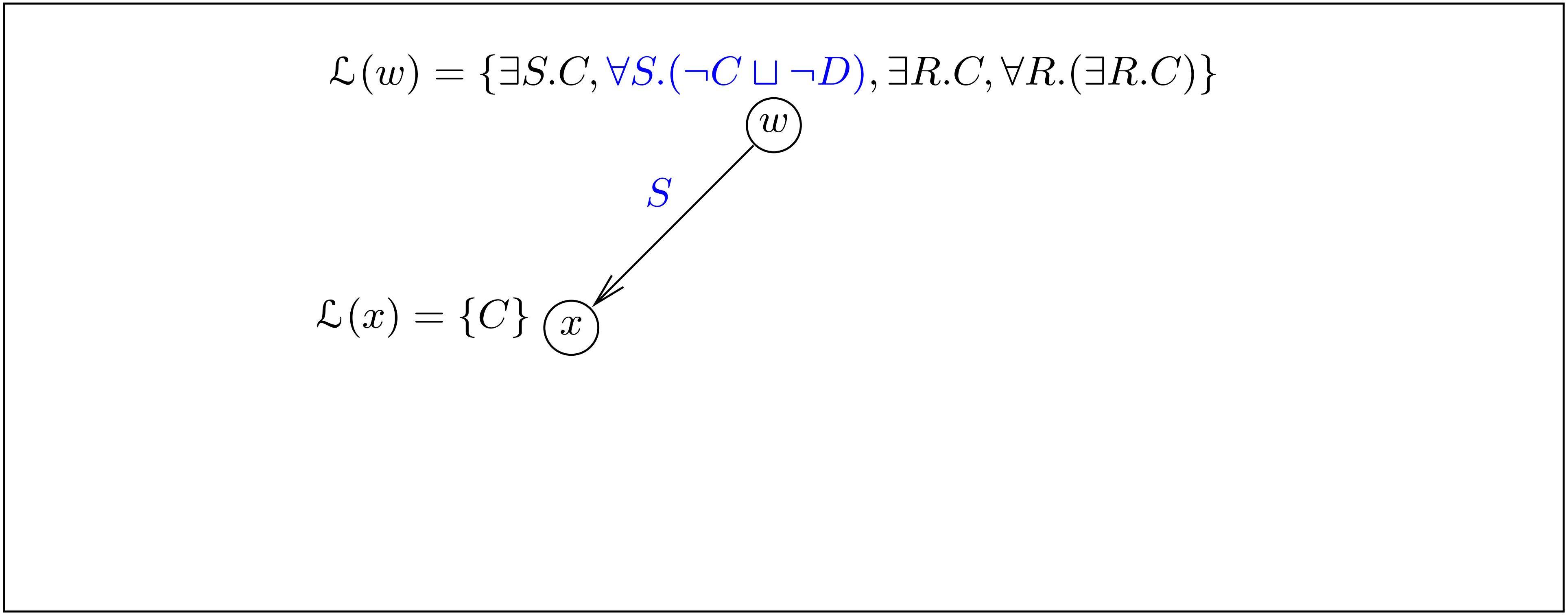
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



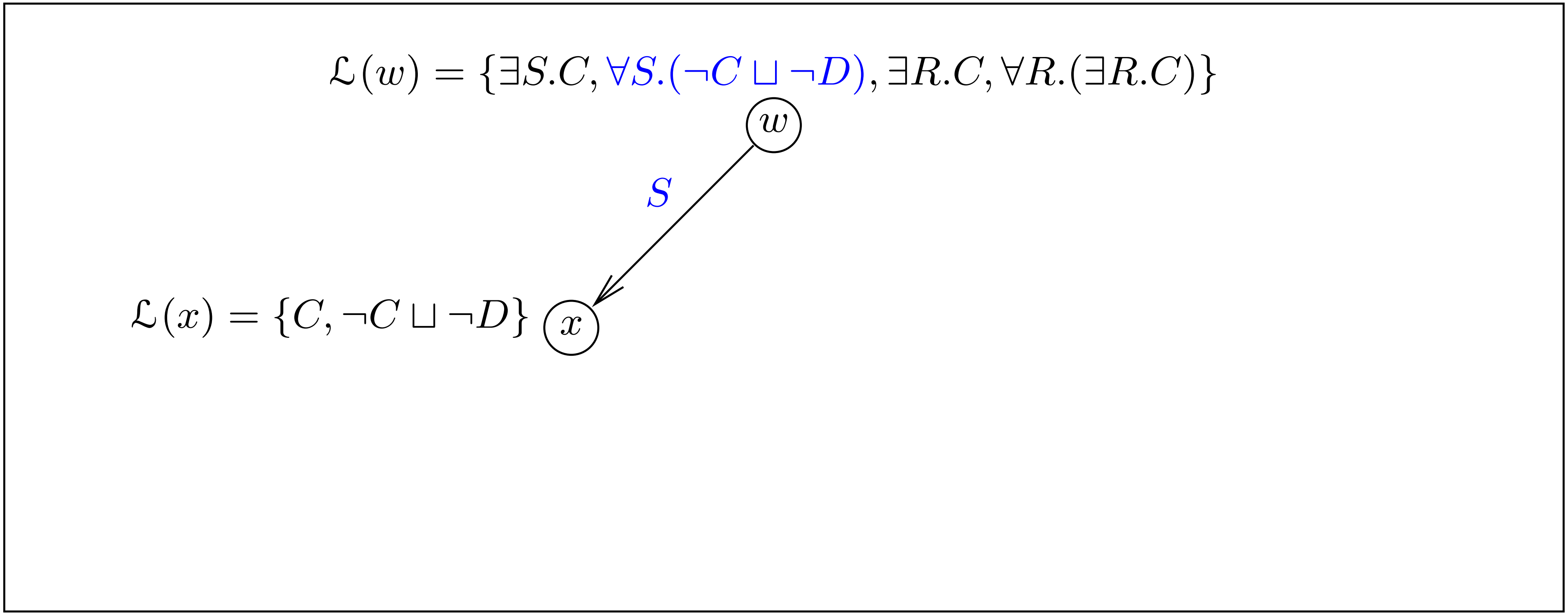
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

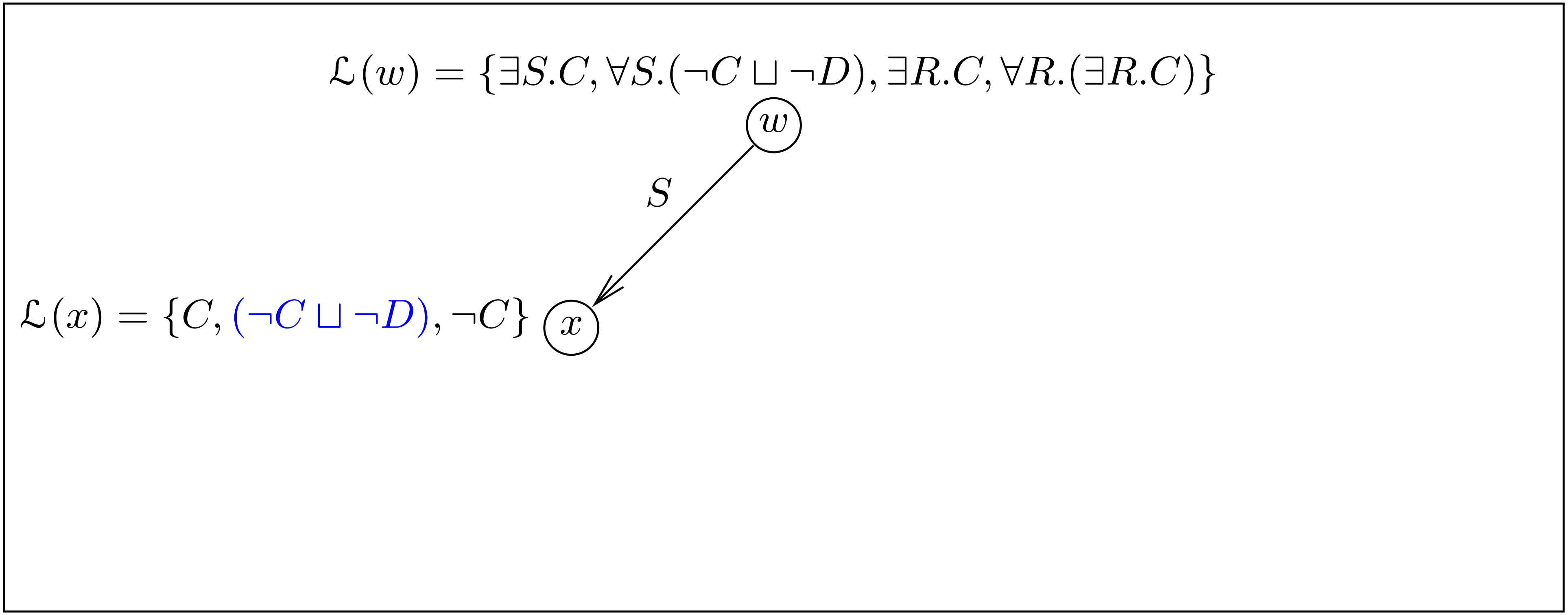
$$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$$

$$\begin{tikzcd} & w \arrow[dl, "S"'] \\ x & \end{tikzcd}$$

$$\mathcal{L}(x) = \{C, \neg C \sqcup \neg D\}$$

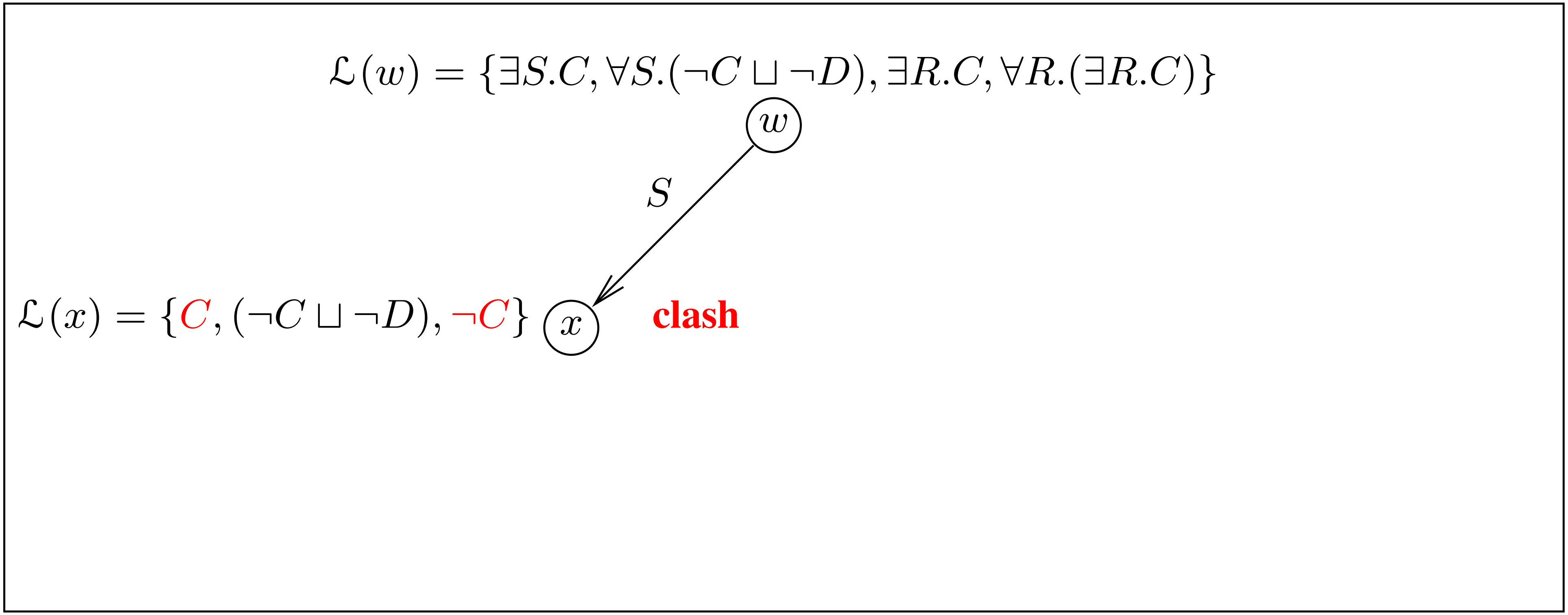
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



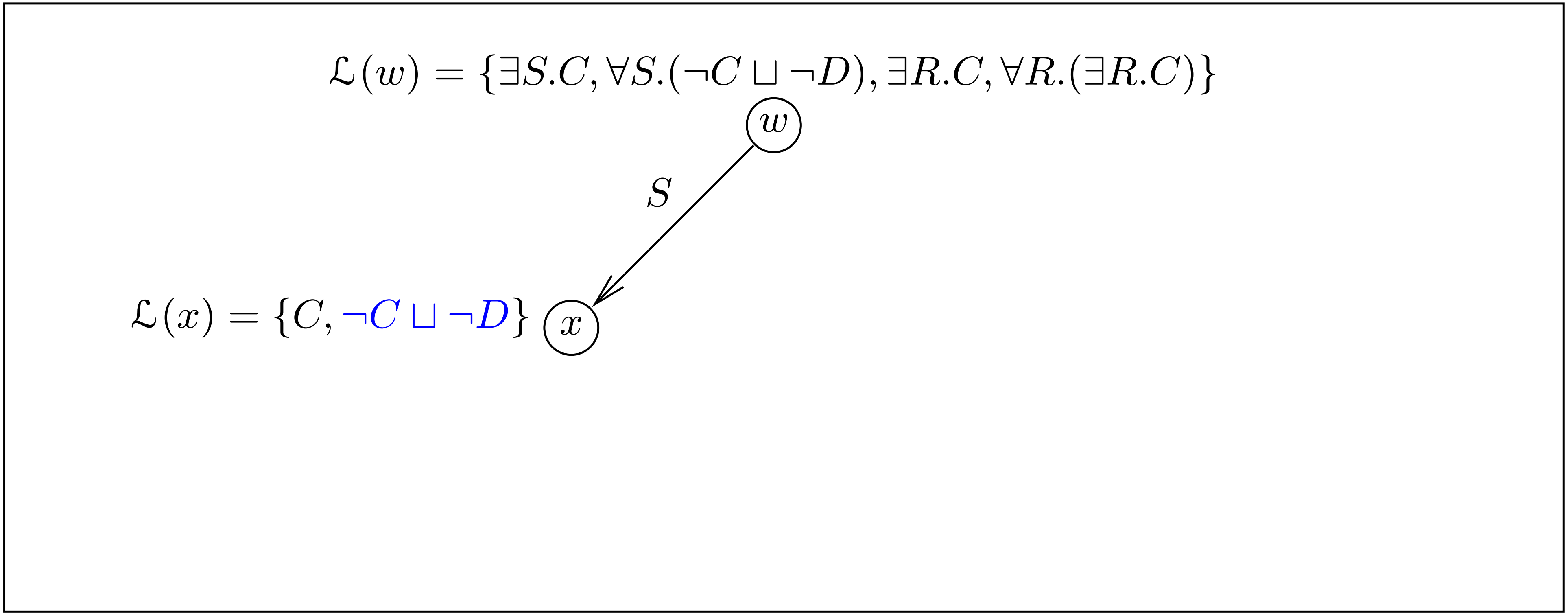
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Tableaux Algorithm — Example

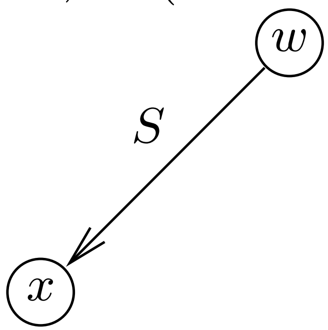
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

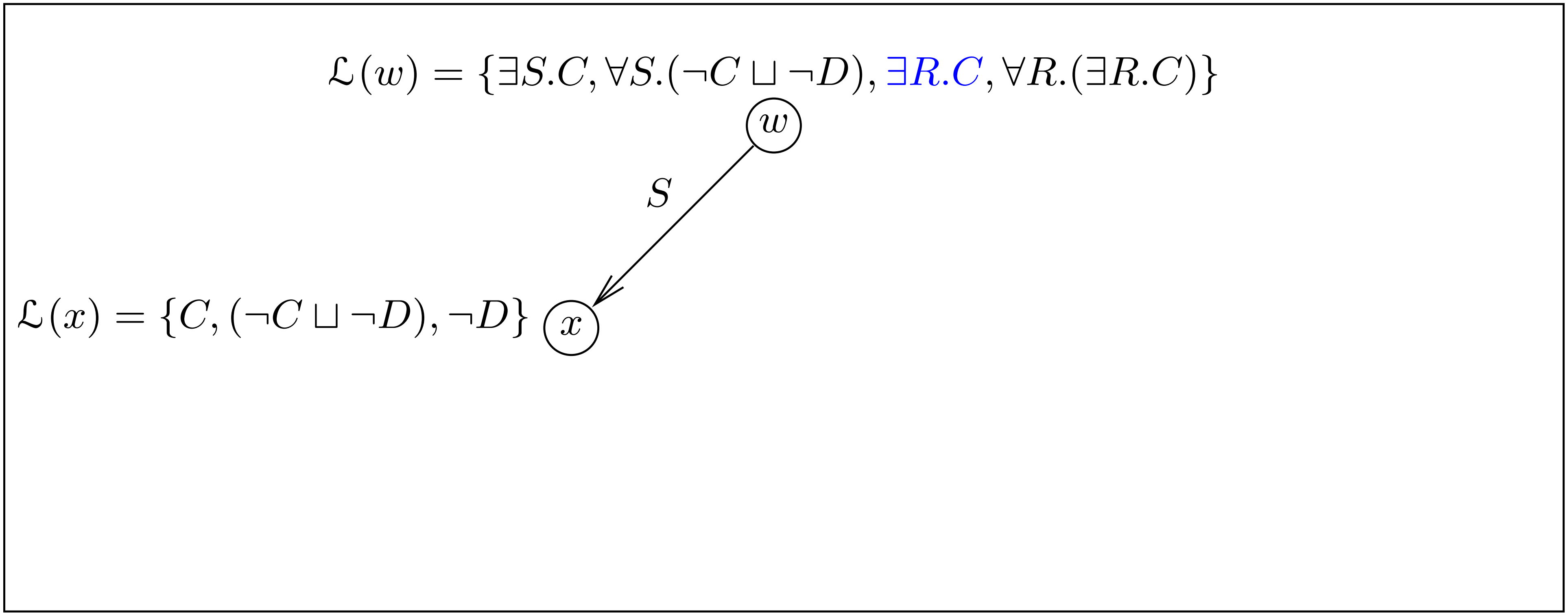
$$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$$



$$\mathcal{L}(x) = \{C, (\neg C \sqcup \neg D), \neg D\}$$

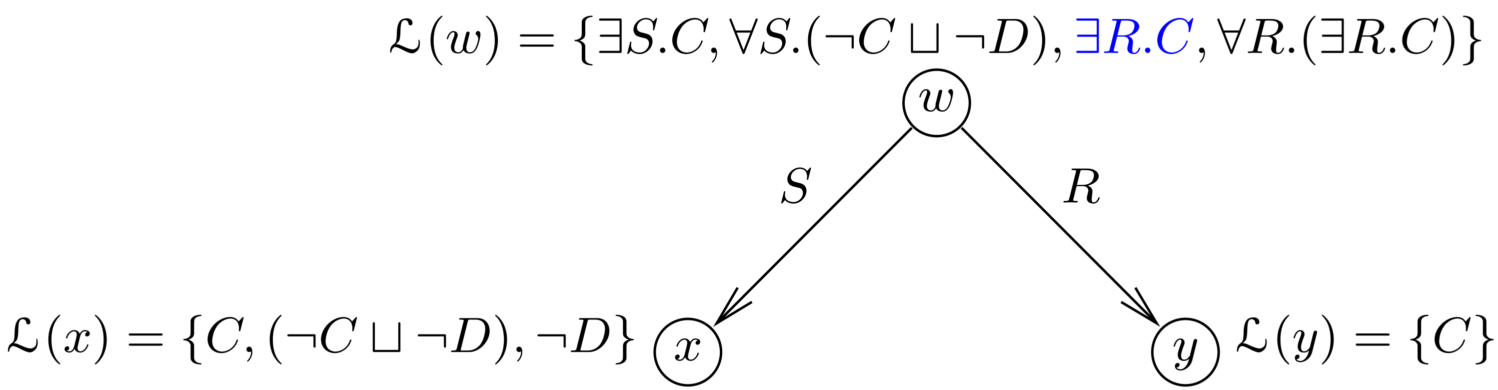
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



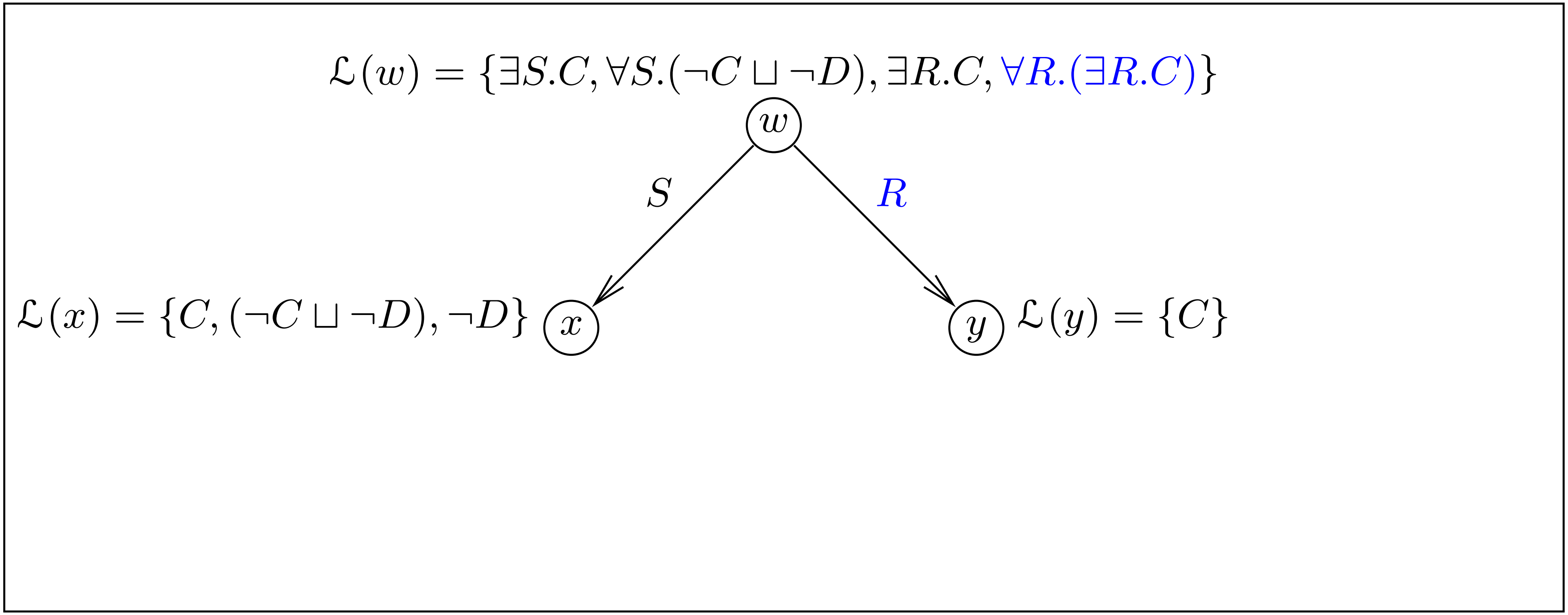
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



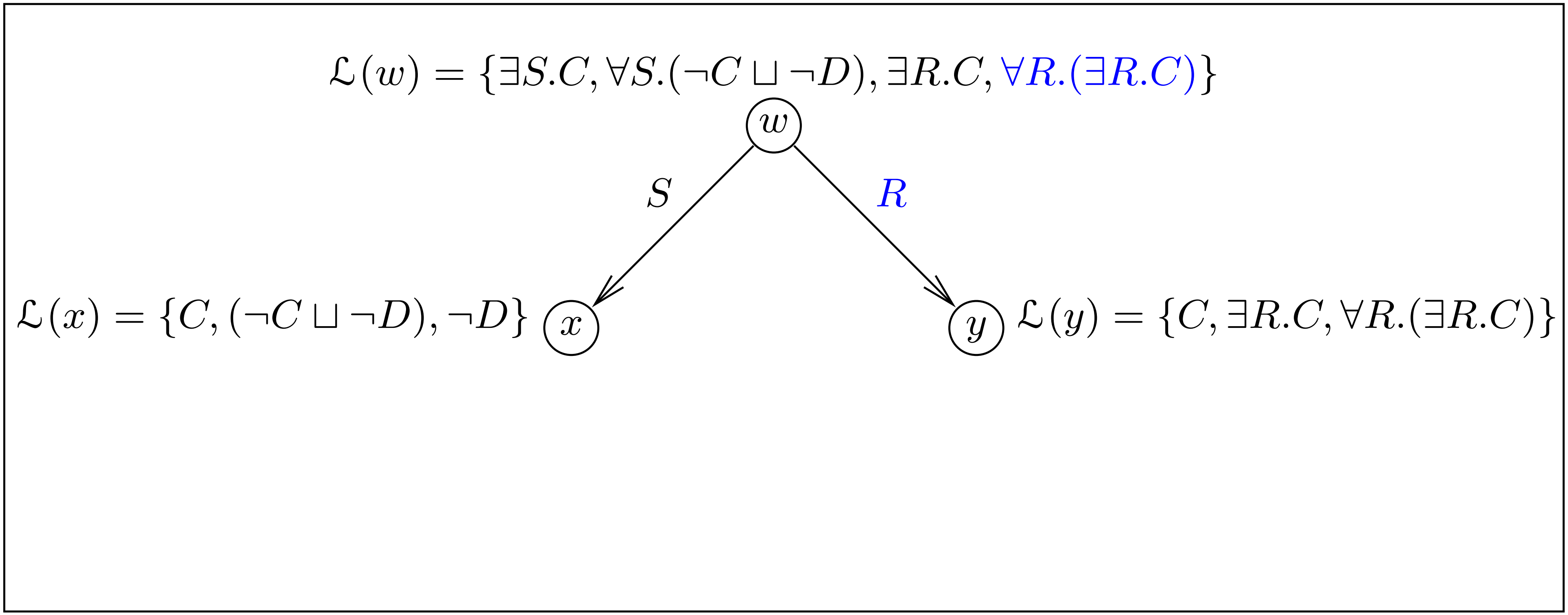
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



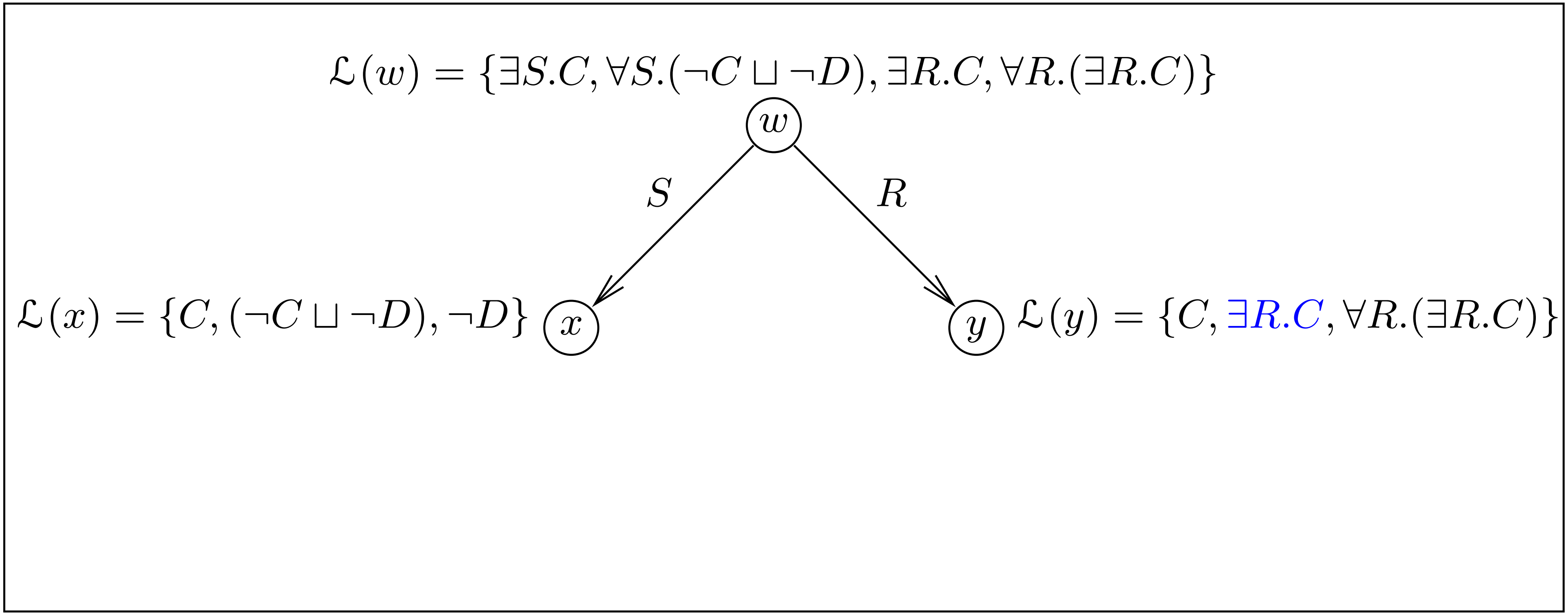
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



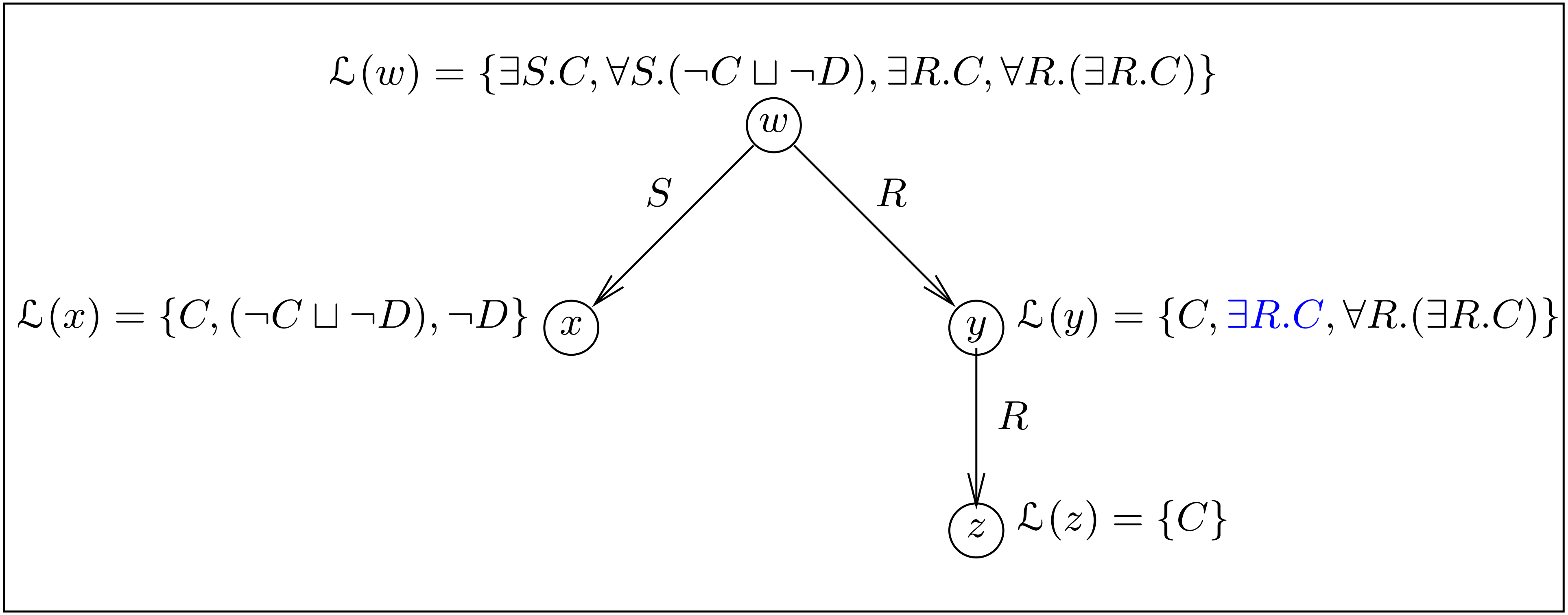
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



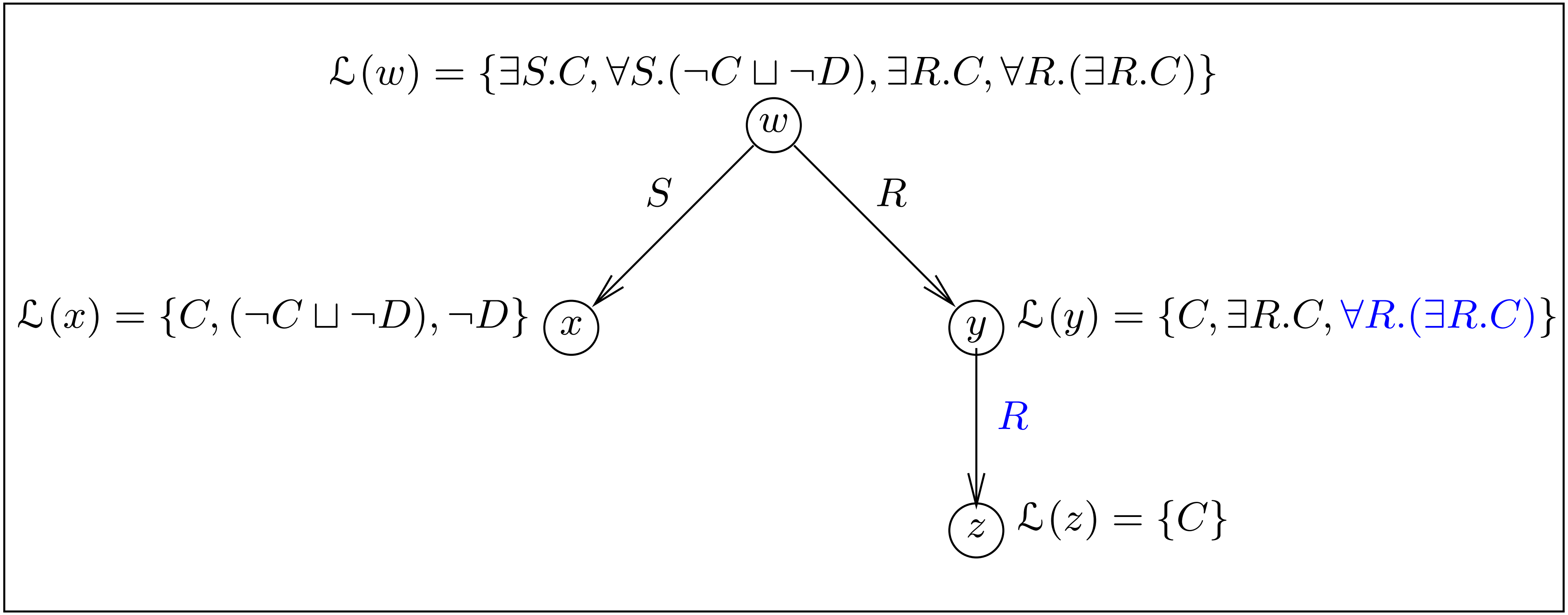
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



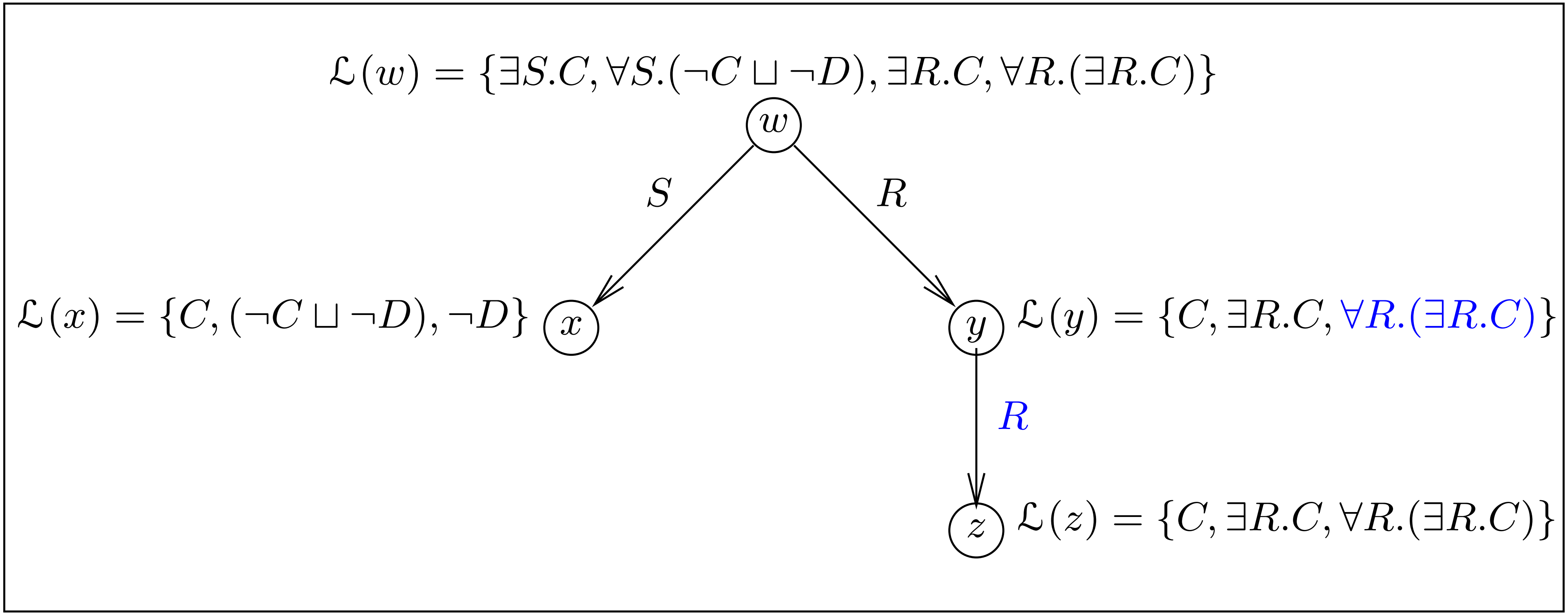
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



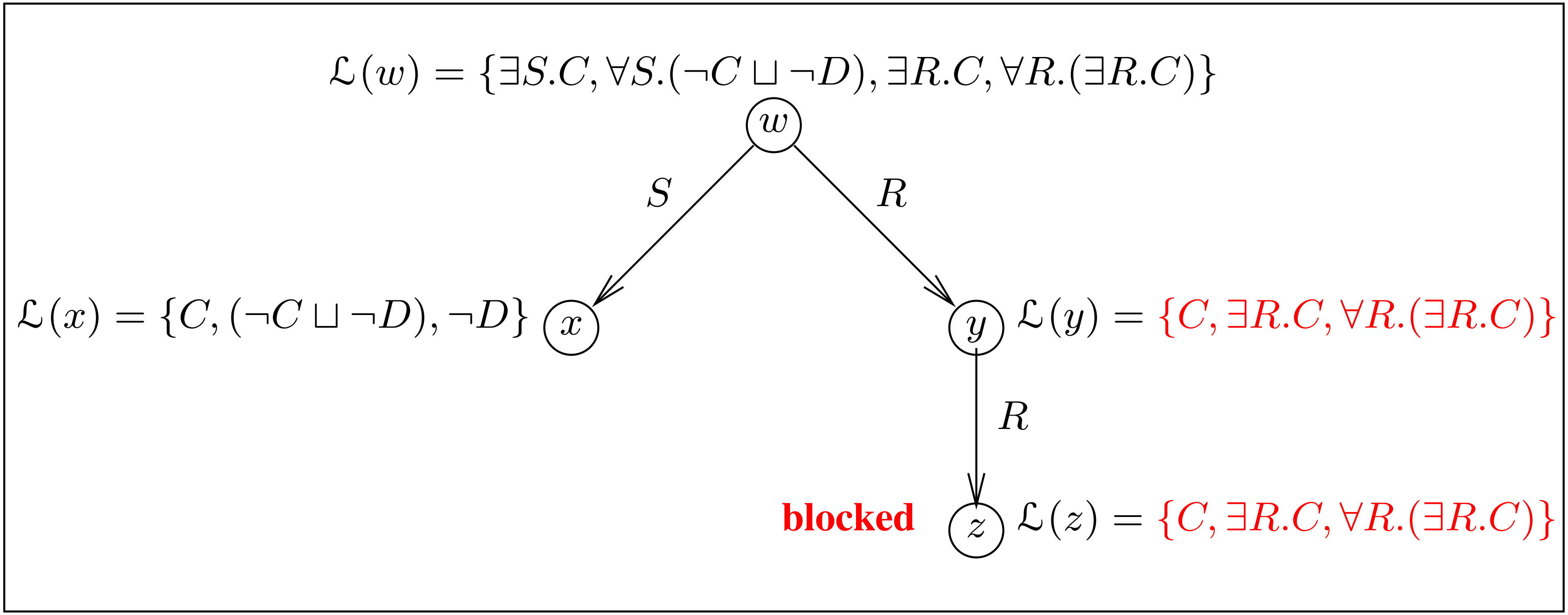
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



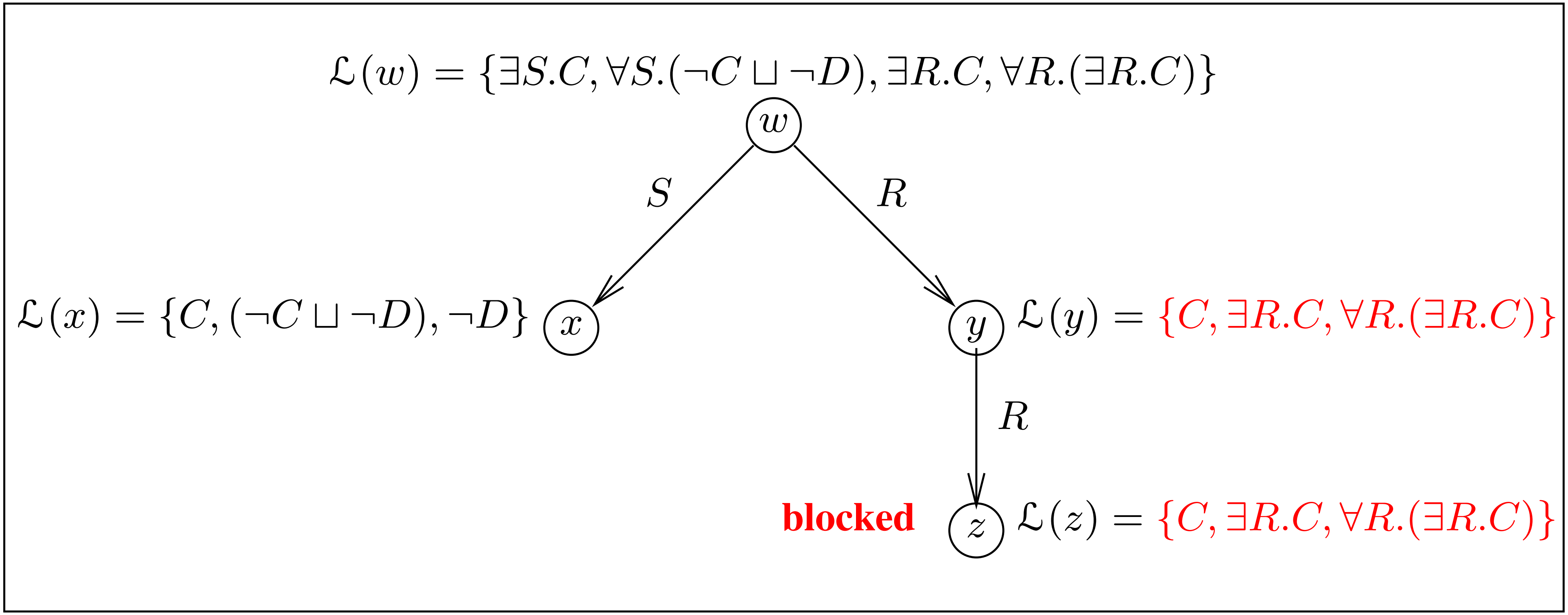
Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Tableaux Algorithm — Example

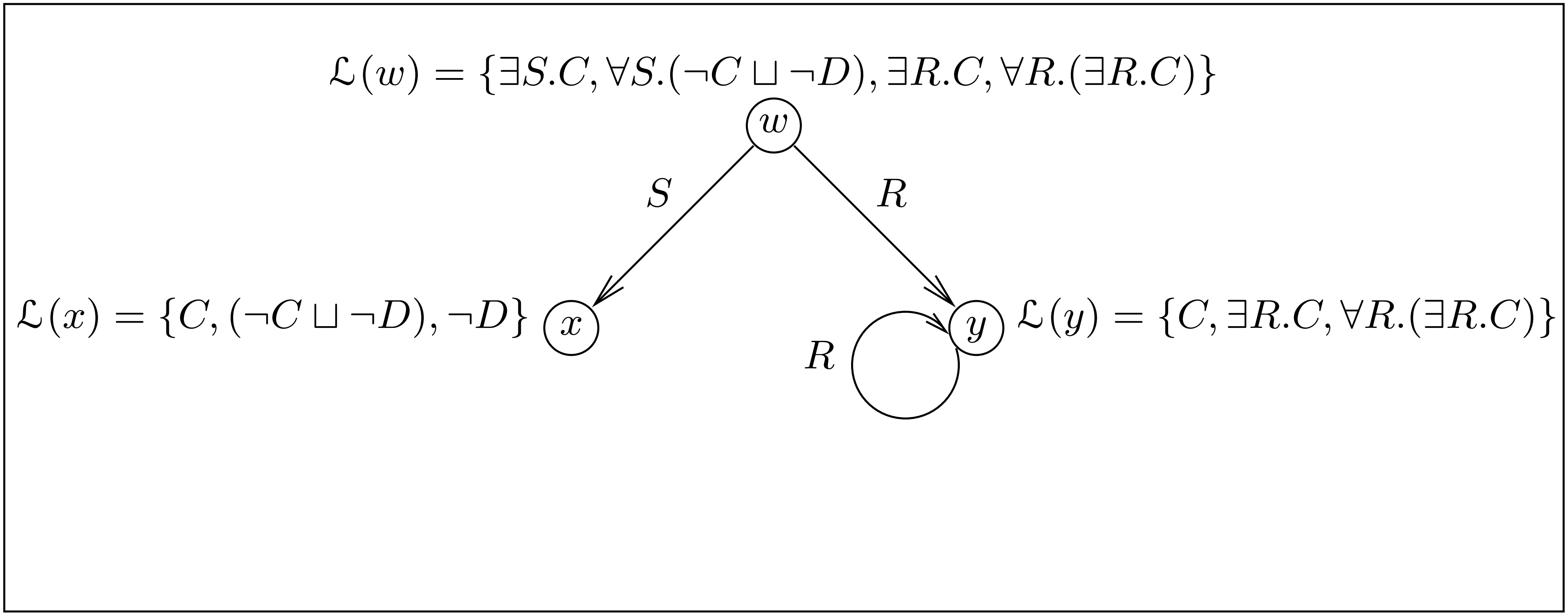
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Concept is **satisfiable**: $\mathbf{T}$ corresponds to **model**

Tableaux Algorithm — Example

Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



Concept is **satisfiable**: $\mathbf{T}$ corresponds to **model**

OWL Example

Develop a sample ontology in the domain of people, pets, vehicles, and newspapers

- **Understand the basic reasoning mechanisms of OWL DL**
  - **Subsumption**
    - **Automatic classification: an ontology built collaboratively**
    - **Instance classification**
    - **Detecting redundancy**
  - **Consistency checking: unsatisfiable restrictions in a Tbox (are the classes coherent?)**

Interesting results (I). Automatic classification

And old lady is a person who is elderly and female.
Old ladies must have some animal as pets and all their pets are cats.

$$elderly \subseteq person \cap adult$$

$$woman \equiv person \cap female \cap adult$$

$$catOwner \equiv person \cap \exists hasPet.cat$$

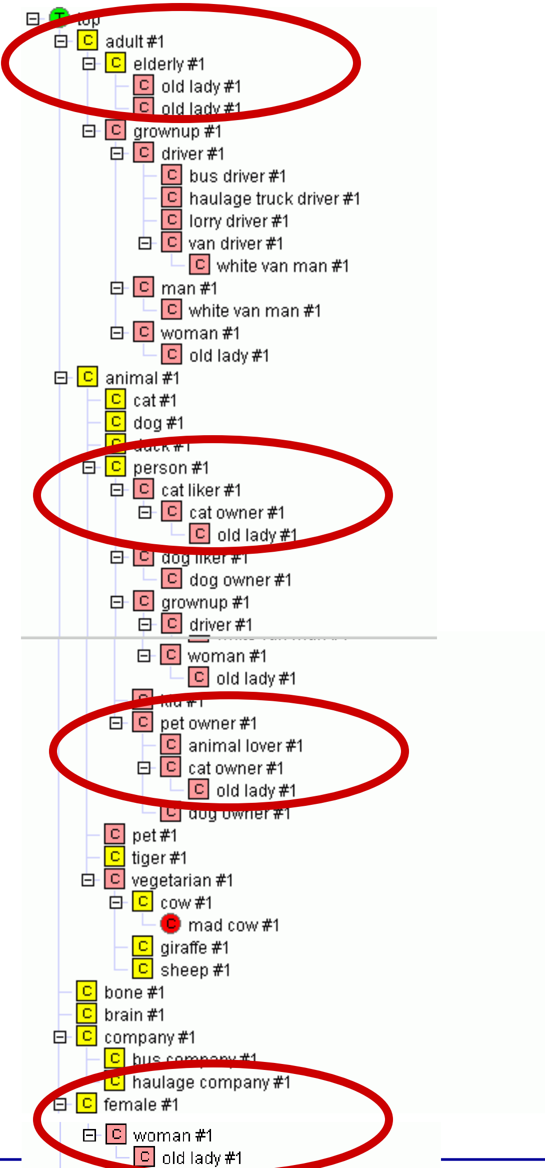
$$oldLady \equiv person \cap female \cap elderly$$

$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$

We obtain:

Old ladies must be women.
Every old lady must have a pet cat
Hence, every old lady must be a cat owner

$$oldLady \subseteq woman \cap elderly \cap catOwner$$



Interesting results (II). Instance classification

A pet owner is a person who has animal pets
Old ladies must have some animal as pets and all their pets are cats.
Has pet has domain person and range animal
Minnie is a female, elderly, who has a pet called Tom.

$$petOwner \equiv person \cap \exists hasPet.animal$$

$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$

$$hasPet \subseteq (person, animal)$$

$$Minnie \in female \cap elderly$$

$$hasPet(Minnie, Tom)$$

We obtain:

Minnie is a person
Hence, Minnie is an old lady
Hence, Tom is a cat

$$Minnie \in person; Tom \in animal$$

$$Minnie \in petOwner$$

$$Minnie \in oldLady$$

$$Tom \in cat$$

Interesting results (III). Instance classification and redundancy detection

An animal lover is a person who has at least three pets
Walt is a person who has pets called Huey, Louie and Dewey.

$$animalLover \equiv person \cap (\geq 3 hasPet)$$

$$Walt \in person$$

$$hasPet(Walt, Huey)$$

$$hasPet(Walt, Louie)$$

$$hasPet(Walt, Dewey)$$

We obtain:

- **Walt is an animal lover**
- **Walt is a person is redundant**

$$Walt \in animalLover$$

Interesting results (IV). Instance classification

A van is a type of vehicle
A driver must be adult
A driver is a person who drives vehicles
A white van man is a man who drives vans and white things
White van mans must read only tabloids
Q123ABC is a white thing and a van
Mick is a male who reads Daily Mirror and drives Q123ABC

$$van \subseteq vehicle$$
$$driver \subseteq adult$$
$$driver \equiv person \cap \exists drives.vehicle$$
$$whiteVanMan \equiv man \cap \exists drives.(van \cap whiteThing)$$
$$whiteVanMan \subseteq \forall reads.tabloid$$
$$Q123ABC \in whiteThing \cap van$$
$$Mick \in male$$
$$reads(Mick, DailyMirror)$$
$$drives(Mick, Q123ABC)$$

We obtain:

Mick is an adult
Mick is a white van man
Daily Mirror is a tabloid

$$Mick \in adult$$
$$Mick \in whiteVanMan$$
$$DailyMirror \in tabloid$$

Interesting results (V). Consistency checking

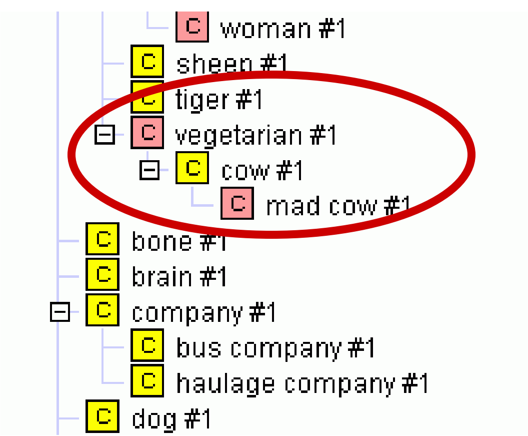
Cows are vegetarian.
A vegetarian is an animal that does not eat animals nor parts of animals.
A mad cow is a cow that eats brains that can be part of a sheep

$$cow \subseteq vegetarian$$

$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap \forall eats.\neg(\exists partOf.animal))$$

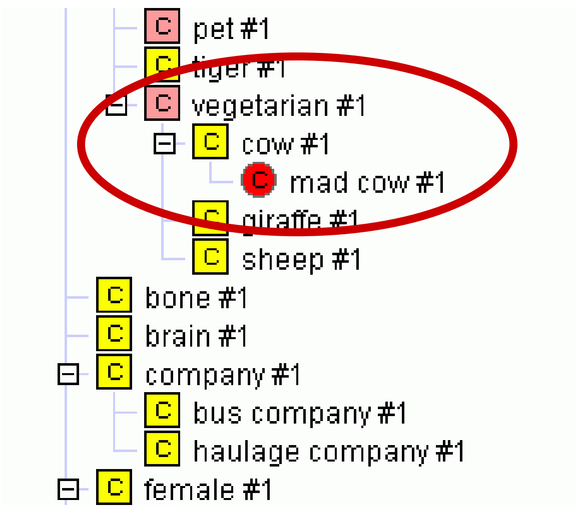
$$madCow \equiv cow \cap \exists eats.(brain \cup \exists partOf.sheep)$$

$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \perp$$



We obtain:

Mad cow is unsatisfiable



OWL Example

When to use a classifier

1. **At author time (pre-coordination): As a compiler**
   - Ontologies will be delivered as "pre-coordinated" ontologies to be used without a reasoner
   - To make extensions and additions quick, easy, and responsive, distribute developments, empower users to make changes
   - Part of an ontology life cycle

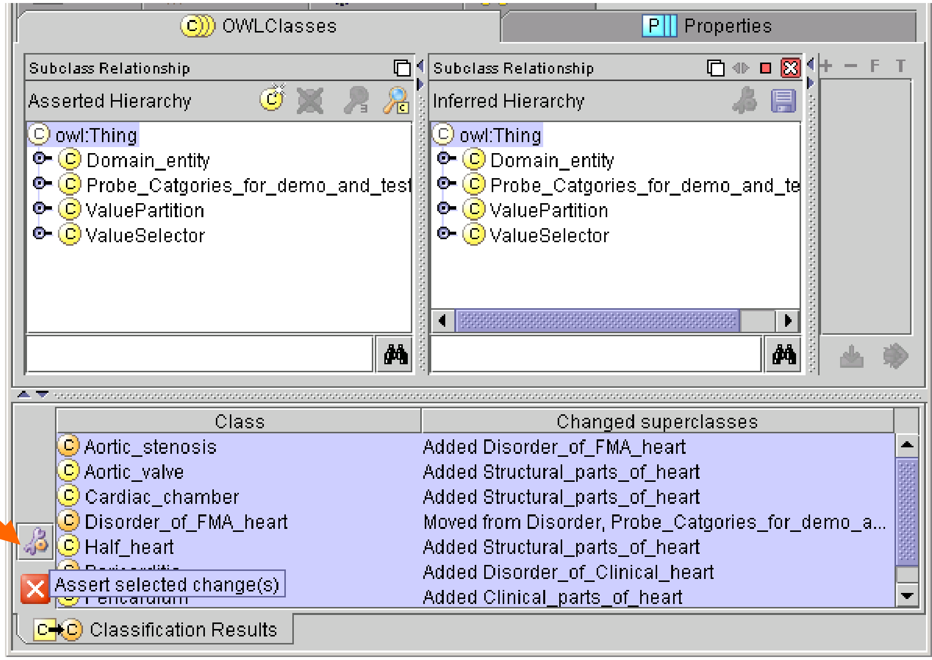
2. **At delivery time (post-coordination): as a normalisation service**
   - Many fixed ontologies are too big and too small
     - Too big to find things; too small to contain what you need
       - Create them on the fly
   - Part of an ontology service

3. **At application time (inference): as a reasoner**
   - Decision support, query optimisation, schema integration, etc.
   - Part of a reasoning service

1. Pre-coordinated delivery: classifier as compiler

- **Develop an ontology**
  - A classifier can be used to detect and correct inconsistencies
- **Classify the ontology**
- **Commit classifier results to a pre-coordinated ontology**

Assert ("Commit") changes inferred by classifier



- **Deliver it**
  - In OWL-Lite or RDFS
- **Use RDQL, SPARQL, or your favourite RDF(S) query tool**

2. Post Coordination: classifier as a service

- **Logic based ontologies act as a conceptual lego**
  - Modularisation/Normalisation is needed to make them easier to maintain

hand

extremity

body

chronic

acute

abnormal
normal

ischaemic

deletion

polymorphism

gene

protein

cell

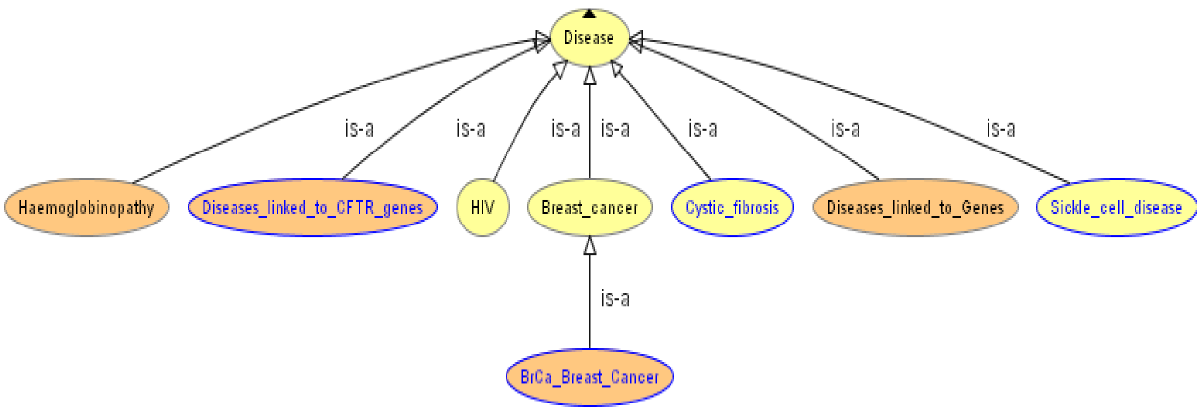
expression

Lung
inflammation
infection

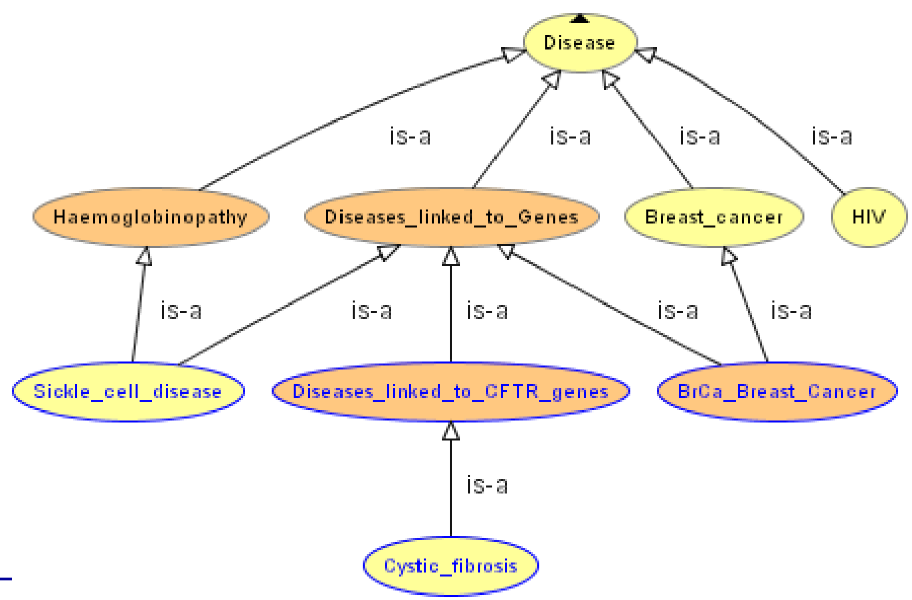
bacterial

2. Post Coordination. Example (I)

- **Build a simple tree**

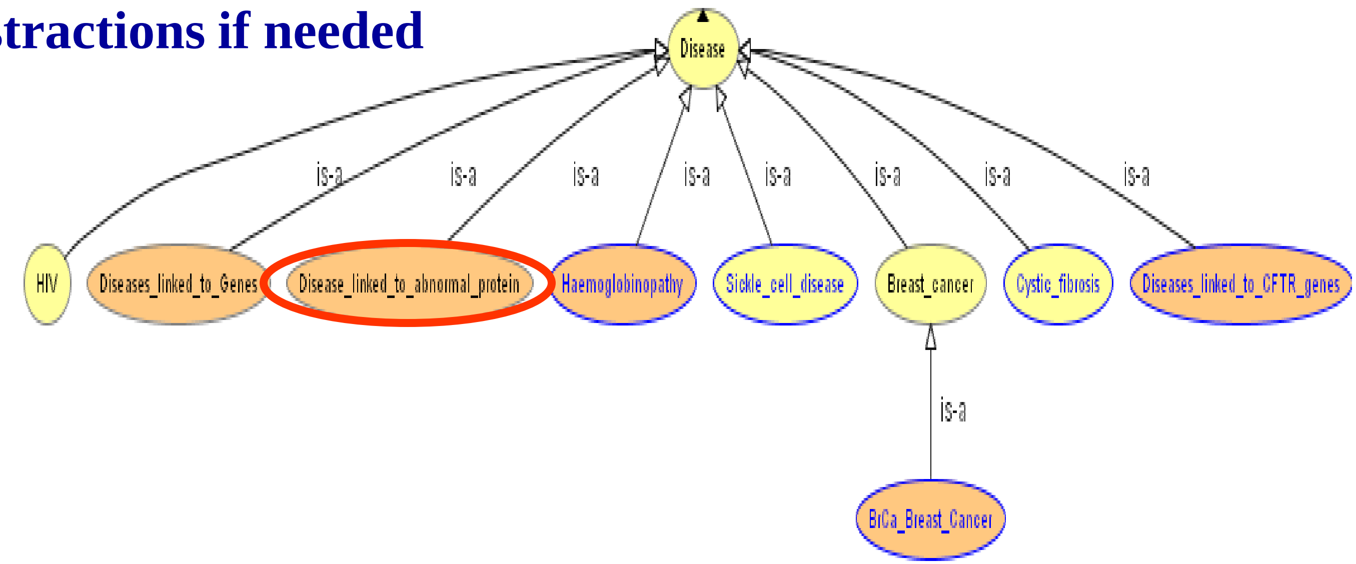


- **Let the classifier organise it and check consistency**

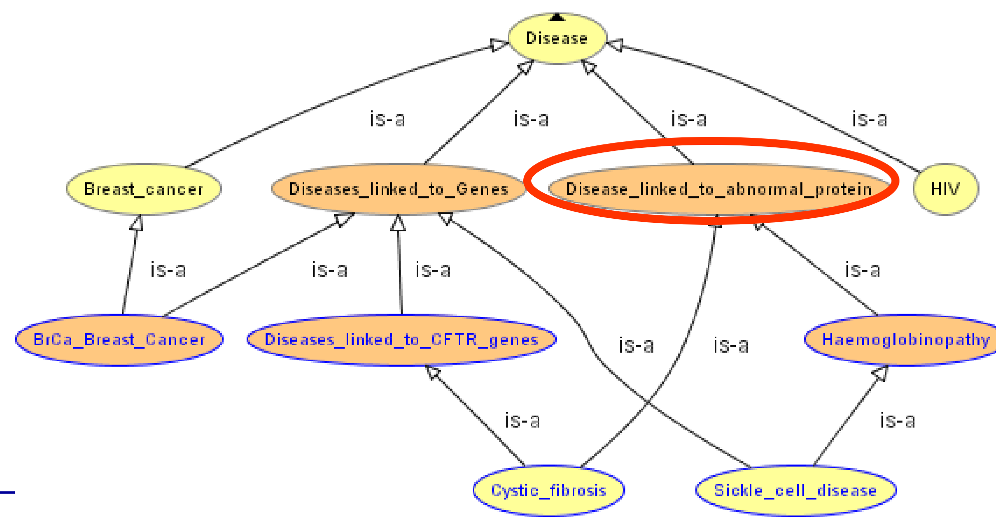


2. Post Coordination. Example (II)

- **Add more abstractions if needed**



- **Let the classifier organise it again and check consistency!!**



3. Inference at application run-time

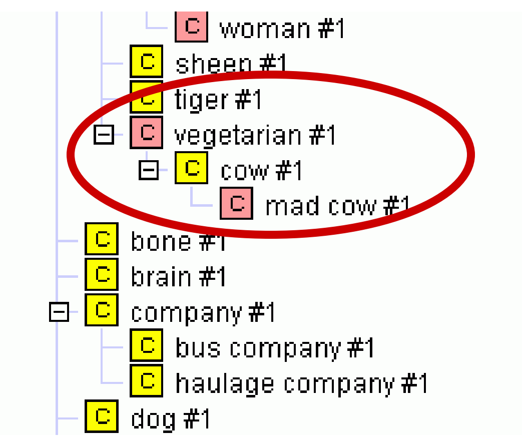
Cows are vegetarian.
A vegetarian is an animal that does not eat animals nor parts of animals.
A mad cow is a cow that eats brains that can be part of a sheep

$$cow \subseteq vegetarian$$

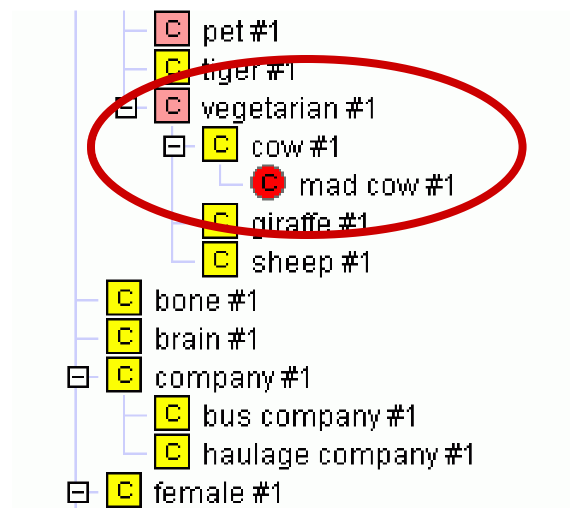
$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap \forall eats.\neg(\exists partOf.animal))$$

$$madCow \equiv cow \cap \exists eats.(brain \cup \exists partOf.sheep)$$

$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \perp$$



We obtain:
Mad cow is unsatisfiable



OWL Example

Develop a sample ontology in the domain of people, pets, vehicles, and newspapers

- Understand the basic reasoning mechanisms of OWL DL

    Subsumption

        Automatic classification: an ontology built collaboratively

        Instance classification

        Detecting redundancy

    Consistency checking: unsatisfiable restrictions in a Tbox (are the classes coherent?)

OWL Classifier limitations

- **Numbers and strings**
  - Simple concrete data types in spec
  - User defined XML data types enmeshed in standards disputes
  - No standard classifier deals with numeric ranges
    - Although several experimental ones do

- **is-part-of and has-part**
  - Totally doubly-linked structures scale horridly

- **Handling of individuals**
  - Variable with different classifiers
  - oneOf works badly with all classifiers at the moment

Ontology languages

Oscar Corcho
ocorcho@fi.upm.es
http://www.oeg-upm.net/

Ontological Engineering Group
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain