# Table of Contents

Unit 1
1. What is an ontology?
   1.1 The Role of Ontologies in the Semantic Web
   1.2 Theoretical Foundations of Ontologies

Unit 2
2. How can we build ontologies? Methods, techniques and methodologies

Unit 3
3. How can we implement ontologies? Ontology languages

Unit 4
4. How can we use ontologies? Reasoners and ontology APIs

Unit 5
5. How can we build Semantic Web applications?

# How can we implement ontologies? Ontology languages

**Asunción Gómez-Pérez**
**Mariano Fernández-López**
**Oscar Corcho**
asun@fi.upm.es, mfernandez.eps@ceu.es, ocorcho@cs.man.ac.uk
Grupo de Ontologías
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

1. **Knowledge Representation Formalisms**
2. **Frames and semantic networks: RDF and RDF Schema**

   **2.1 RDF and RDF Schema primitives**

   **2.2 Formalisation with RDF(S)**
3. **Description Logic: OWL**

   **3.1 OWL primitives and DL syntax**

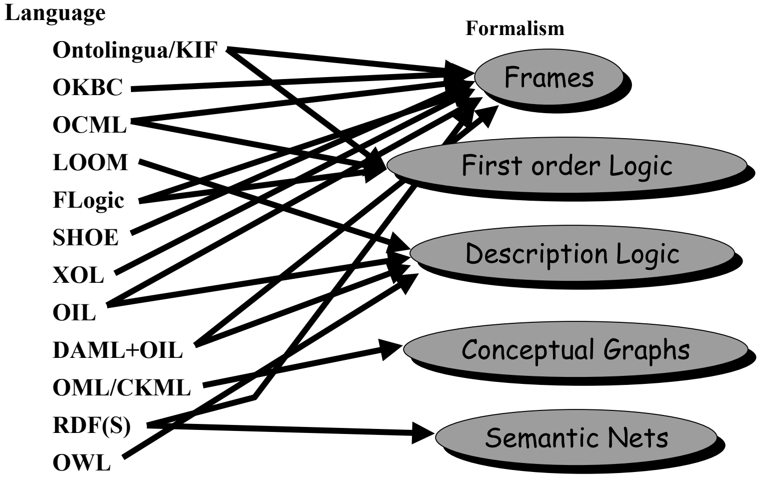
   **3.2 Formalisation with OWL**

   **3.3 The Protégé-OWL plug-in**

   **3.4 Reasoning with OWL. Tableaux algorithms**

   **3.5 When to use a reasoner**

# KR Formalisms



**Language**

**Ontolingua/KIF**

**OKBC**

**OCML**

**LOOM**

**FLogic**

**SHOE**

**XOL**

**OIL**

**DAML+OIL**

**OML/CKML**

**RDF(S)**

**OWL**

**Formalism**

Frames

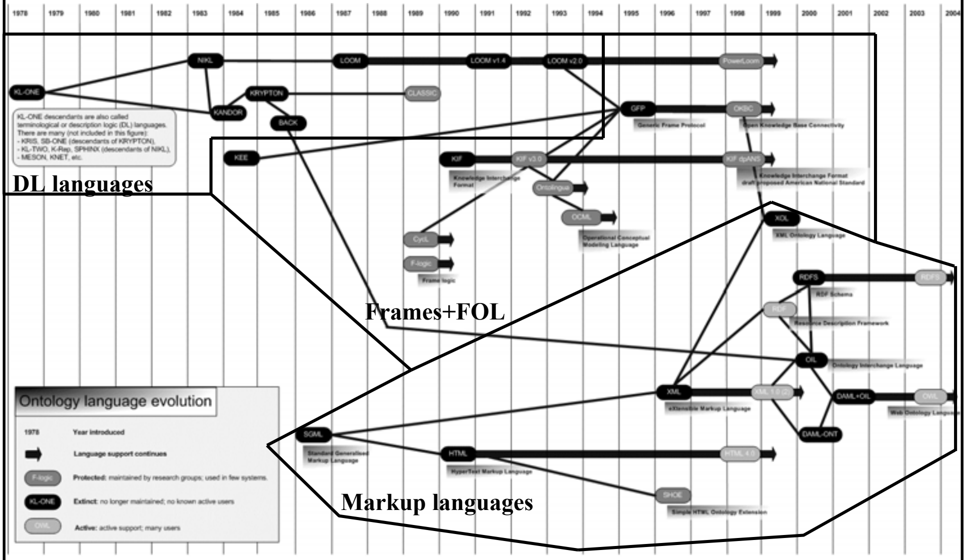
First order Logic

Description Logic

Conceptual Graphs

Semantic Nets

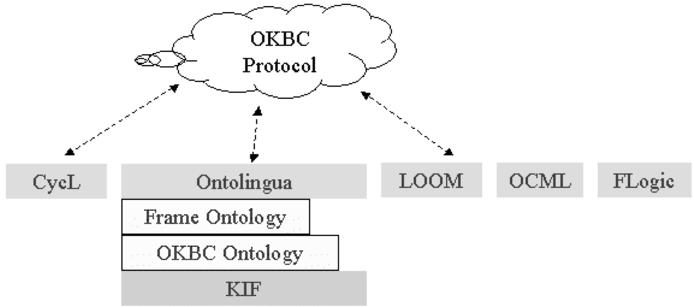
# Ontology language evolution


1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
KL-ONE
NIKL
KANDOR
KRYPTON
BACK
LOOM
LOOM v1.4
LOOM v2.0
PowerLoom
CLASSIC
KL-ONE descendants are also called terminological or description logic (DL) languages.
There are many (not included in this figure):
- KRIS, SB-ONE (descendants of KRYPTON),
- KL-TWO, K-Rep, SPHINX (descendants of NIKL),
- MESON, KNET, etc.
DL languages
KEE
GFP
Generic Frame Protocol
OKBC
Open Knowledge Base Connectivity
KIF
Knowledge Interchange Format
KIF v3.0
KIF dpANS
Knowledge Interchange Format draft proposed American National Standard
Ontolingua
OCML
Operational Conceptual Modeling Language
CycL
F-logic
Frame logic
Frames+FOL
XOL
XML Ontology Language
RDFS
RDF Schema
RDF
Resource Description Framework
OIL
Ontology Interchange Language
XML
eXtensible Markup Language
XML 1.0 (2)
DAML+OIL
OWL
Web Ontology Language
DAML-ONT
SGML
Standard Generalised Markup Language
HTML
HyperText Markup Language
HTML 4.0
SHOE
Simple HTML Ontology Extension
Markup languages
Ontology language evolution
1978
Year introduced
Language support continues
F-logic
Protected: maintained by research groups; used in few systems.
KL-ONE
Extinct: no longer maintained; no known active users
OWL
Active: active support; many users

# 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**

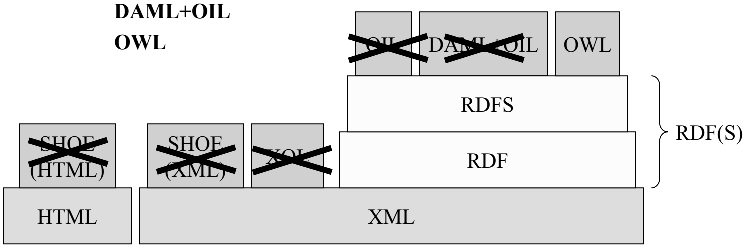
**SHOE**

**XOL**

**OIL**

**DAML+OIL**

**OWL**



## Table of Contents

1. **Knowledge Representation Formalisms**
2. **Frames and semantic networks: RDF and RDF Schema**

    **2.1 RDF and RDF Schema primitives**

    **2.2 Formalisation with RDF(S)**

3. **Description Logic: OWL**

    **3.1 OWL primitives and DL syntax**

    **3.2 Formalisation with OWL**

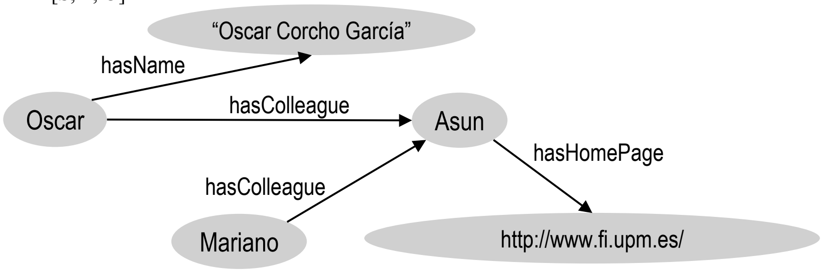
    **3.3 The Protégé-OWL plug-in**

    **3.4 Reasoning with OWL. Tableaux algorithms**

    **3.5 When to use a reasoner**

# RDF: Resource Description Framework

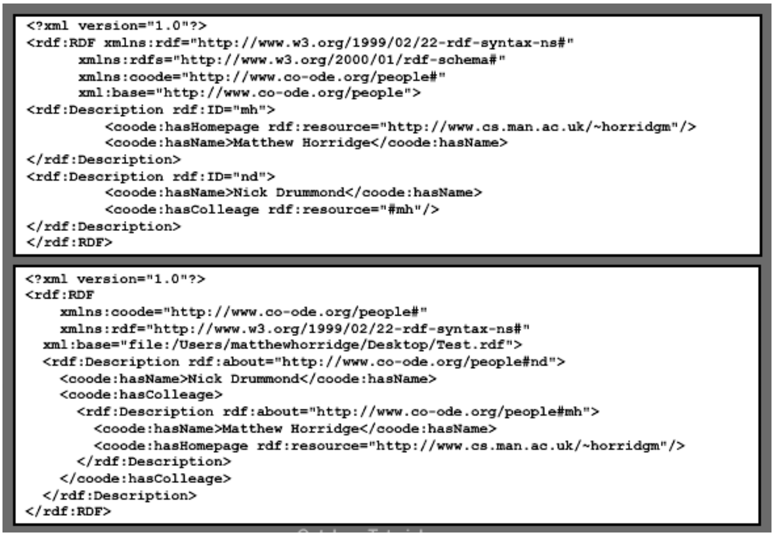
- **W3C recommendation (http://www.w3.org/RDF)**
- **RDF is graphical formalism ( + XML syntax + semantics)**
  - for representing metadata
  - for describing the semantics of information in a machine- accessible way
- **RDF is a basic ontology language**
  - Resources are described in terms of properties 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

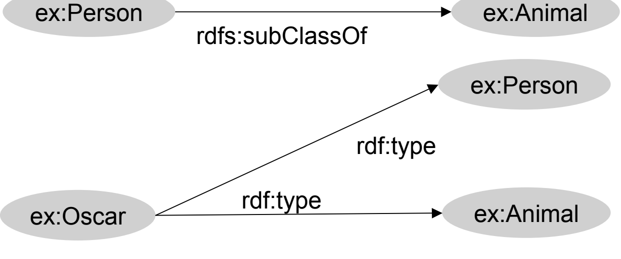


```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
      xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
      xmlns:coode="http://www.co-ode.org/people#"
      xml:base="http://www.co-ode.org/people">
<rdf:Description rdf:ID="mh">
         <coode:hasHomepage 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.:**
  - ∀x,y,z type(x,y) and subClassOf(y,z) → 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

# 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 described in the previous day 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**

## Table of Contents

1. **Knowledge Representation Formalisms**
2. **Frames and semantic networks: RDF and RDF Schema**

   **2.1 RDF and RDF Schema primitives**

   **2.2 Formalisation with RDF(S)**
3. **Description Logic: OWL**

   **3.1 OWL primitives and DL syntax**

   **3.2 Formalisation with OWL.**

   **3.3 The Protégé-OWL plug-in**

   **3.4 Reasoning with OWL. Tableaux algorithms**

   **3.5 When to use a reasoner**

## 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

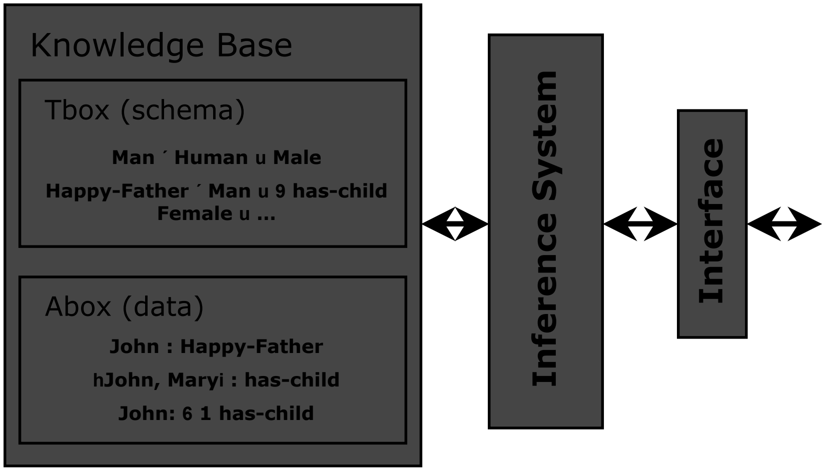
Dean M, Schreiber G. The OWL Web Ontology Language. W3C Recommendation. February 2004.

# 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



Knowledge Base

Tbox (schema)

**Man ´ Human u Male**

**Happy-Father ´ Man u 9 has-child Female u …**

Abox (data)

**John : Happy-Father**

**hJohn, Maryi : has-child**

**John: 6 1 has-child**

**Inference System**

**Interface**

# DL constructors

| Construct | Syntax | Language | | | |
|---|---|---|---|---|---|
| Concept | A | $FL_0$ | $FL^-$ | AL | S[14] |
| Role name | R | | | | |
| Intersection | $C \cap D$ | | | | |
| Value restriction | $\forall$R.C | | | | |
| Limited existential quantification | $\exists$ R | | | | |
| Top or Universal | $\top$ | | | | |
| Bottom | $\bot$ | | | | |
| Atomic negation | $\neg$A | | | | |
| Negation[15] | $\neg$ C | C | | | |
| Union | $C \cup D$ | U | | | |
| Existential restriction | $\exists$ R.C | E | | | |
| Number restrictions | $(\geq n\ R)$ $(\leq n\ R)$ | N | | | |
| Nominals | $\{a_1 \ldots a_n\}$ | O | | | |
| Role hierarchy | $R \subseteq S$ | H | | | |
| Inverse role | $R^-$ | I | | | |
| Qualified number restriction | $(\geq n\ R.C)$ $(\leq n\ R.C)$ | Q | | | |

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 | $C_1 \wedge \ldots \wedge C_n$ |
| unionOf | $C_1 \sqcup \ldots \sqcup C_n$ | Doctor $\sqcup$ Lawyer | $C_1 \vee \ldots \vee C_n$ |
| complementOf | $\neg C$ | $\neg$Male | $\neg C$ |
| oneOf | $\{x_1\} \sqcup \ldots \sqcup \{x_n\}$ | {john} $\sqcup$ {mary} | $x_1 \vee \ldots \vee x_n$ |
| 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 | $\langle P \rangle_n$ |

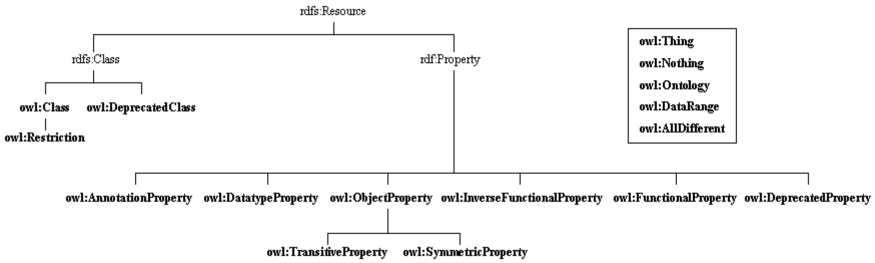
- **XML Schema datatypes are treated as classes**
  - $\forall$hasAge.nonNegativeInteger
- **Nesting of constructors can be arbitrarily complex**
  - Person $\wedge$ $\forall$hasChild.(Doctor $\vee$ $\exists$hasChild.Doctor)
- **Lots of redundancy, e.g., use negations to transform** and **to** or **and**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

# OWL Axioms

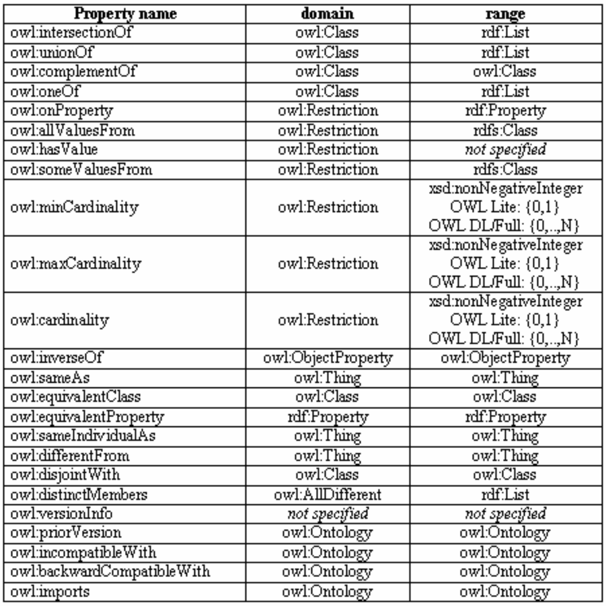
| Axiom | DL Syntax | Example |
|---|---|---|
| subClassOf | $C_1 \sqsubseteq C_2$ | Human $\sqsubseteq$ Animal $\sqcap$ Biped |
| equivalentClass | $C_1 \equiv C_2$ | Man $\equiv$ Human $\sqcap$ Male |
| disjointWith | $C_1 \sqsubseteq \neg C_2$ | Male $\sqsubseteq$ $\neg$Female |
| sameIndividualAs | $\{x_1\} \equiv \{x_2\}$ | {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 (∨)**
  - C ≡ D iff both C ⊆ D and D ⊆ C
  - C disjoint D iff C ⊆ ¬D

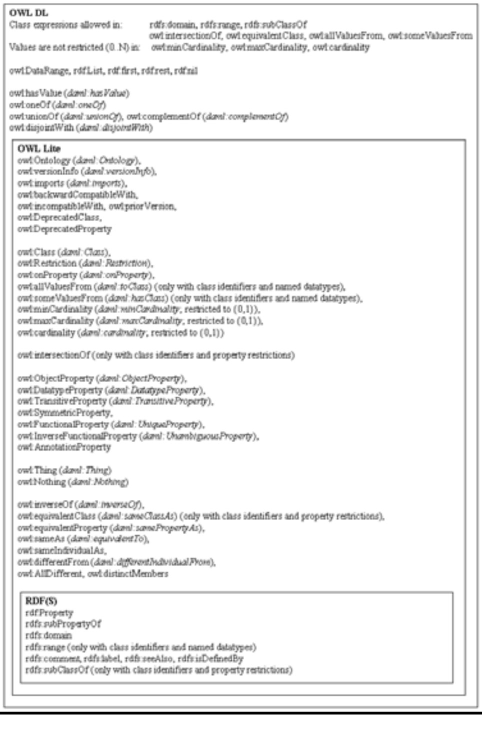
# Class taxonomy of the OWL KR ontology



# 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<br>OWL Lite: {0,1}<br>OWL DL/Full: {0,..,N} |
| owl:maxCardinality | owl:Restriction | xsd:nonNegativeInteger<br>OWL Lite: {0,1}<br>OWL DL/Full: {0,..,N} |
| owl:cardinality | owl:Restriction | xsd:nonNegativeInteger<br>OWL Lite: {0,1}<br>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)

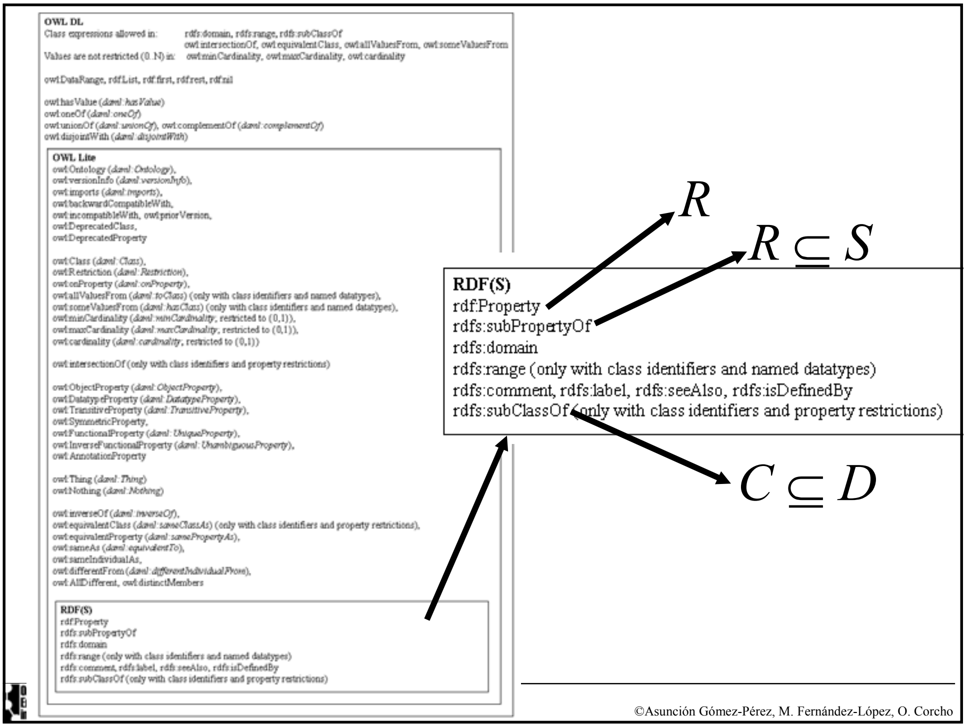
owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*),
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

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*)

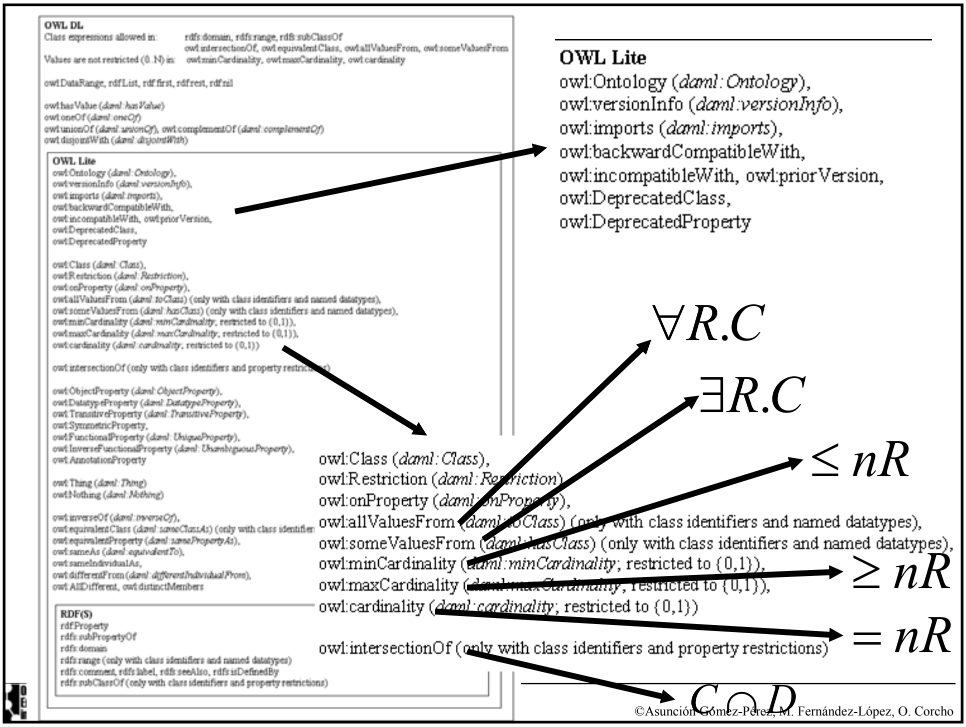
owl:inverseOf (*daml:inverseOf*),
owl:equivalentClass (*daml:sameClassAs*) (only with class identifiers and property restrictions),
owl:equivalentProperty (*daml:samePropertyAs*),
owl:sameAs (*daml:equivalentTo*),
owl:sameIndividualAs,
owl:differentFrom (*daml:differentIndividualFrom*),
owl:AllDifferent,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

**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)

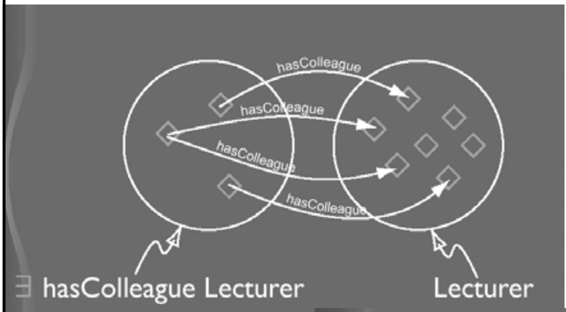
**OWL Lite**

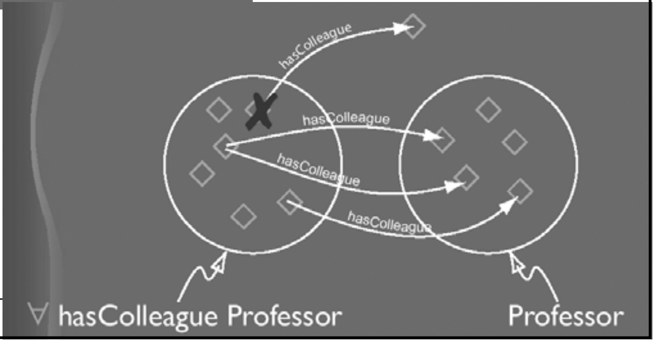
owl:Ontology (*daml:Ontology*),
owl:versionInfo (*daml:versionInfo*),
owl:imports (*daml:imports*),
owl:backwardCompatibleWith,
owl:incompatibleWith, owl:priorVersion,
owl:DeprecatedClass,
owl:DeprecatedProperty

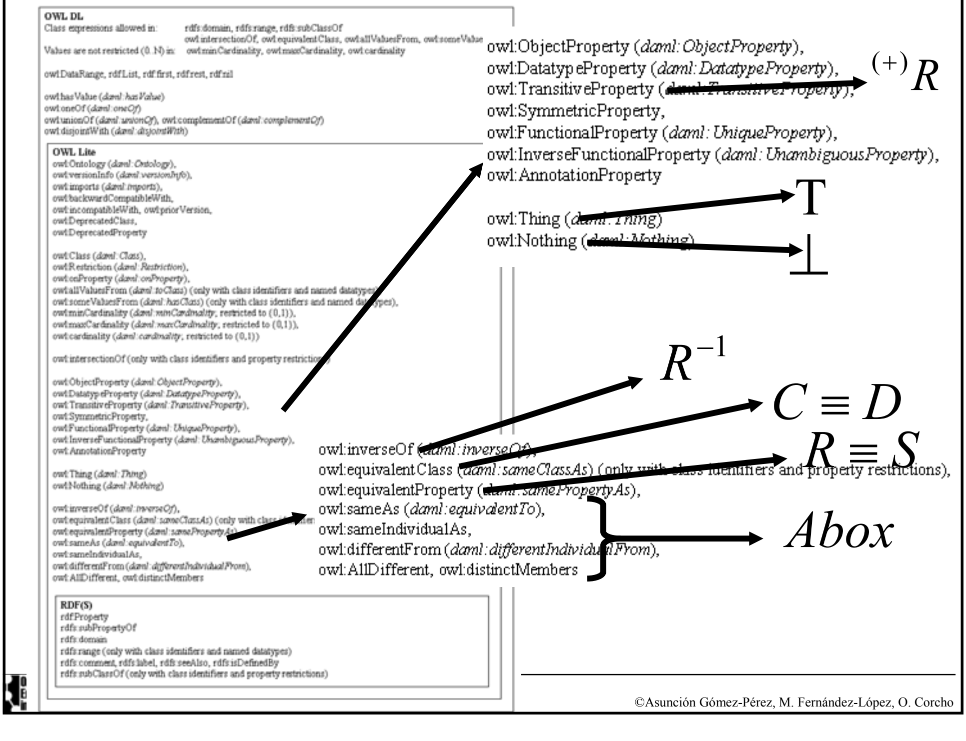
owl:Class (*daml:Class*),
owl:Restriction (*daml:Restriction*),
owl:onProperty (*daml:onProperty*),
owl:allValuesFrom (*daml:toClass*) (only with class identifiers and named datatypes),
owl:someValuesFrom (*daml:hasClass*) (only with class identifiers and named datatypes),
owl:minCardinality (*daml:minCardinality*, 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)

## 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
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)

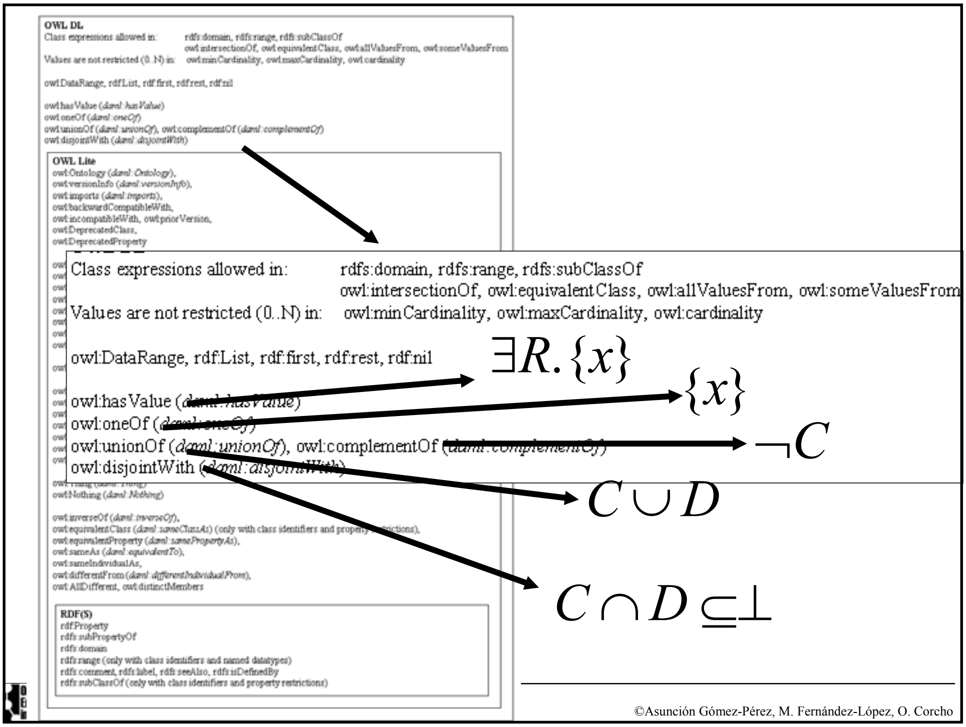
owl:ObjectProperty (*daml:ObjectProperty*),
owl:DatatypeProperty (*daml:DatatypeProperty*),
owl:TransitiveProperty (*daml:TransitiveProperty*), $^{(+)}R$
owl:SymmetricProperty,
owl:FunctionalProperty (*daml:UniqueProperty*),
owl:InverseFunctionalProperty (*daml:UnambiguousProperty*),
owl:AnnotationProperty

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

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

*Abox*

©Asunción Gómez-Pérez, M. Fernández-López, O. Corcho



**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: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)

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

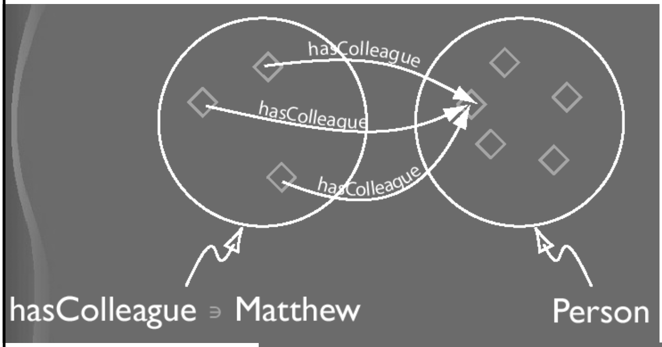
owl:hasValue (*daml:hasValue*) $\rightarrow \exists R.\{x\}$

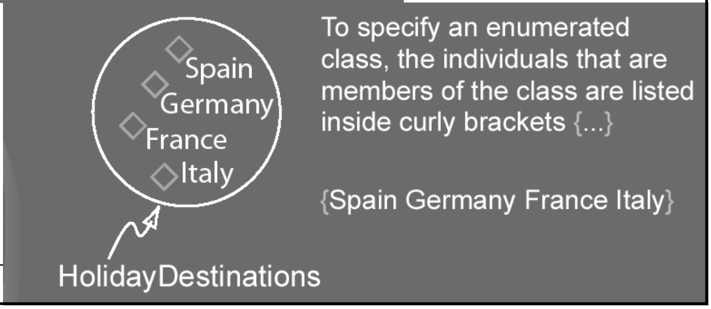
owl:oneOf (*daml:oneOf*) $\rightarrow \{x\}$

owl:unionOf (*daml:unionOf*) $\rightarrow C \cup D$, owl:complementOf (*daml:complementOf*) $\rightarrow \neg C$

owl:disjointWith (*daml:disjointWith*) $\rightarrow C \cap D \subseteq \bot$

## hasValue and oneOf

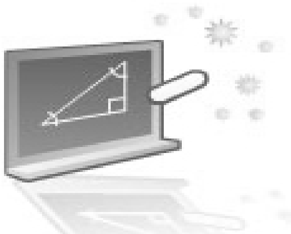




To specify an enumerated class, the individuals that are members of the class are listed inside curly brackets {...}

{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** → $X \subseteq \neg Y$
- **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.**<br>
**The42 is a bus whose service number is 42.**<br>
**Mick is a male who read Daily Mirror and drives Q123ABC.**

# Chunk 1. Formalisation in DL

$$grass \subseteq plant$$

$$tree \subseteq plant$$

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

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

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

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

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

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

## Chunk 2. Formalisation in DL

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

$busCompany \subseteq company; haulageCompany \subseteq company$

$elderly \subseteq person \cap adult$

$kid \equiv person \cap young$

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

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

$grownUp \equiv person \cap adult$

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

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

$young \cap adult \subseteq \bot$

$hasMother \subseteq hasParent$

$hasFather \subseteq hasParent$

## Chunk 3. Formalisation in DL

$magazine \subseteq publication$

$broadsheet \subseteq newspaper$

$tabloid \subseteq newspaper$

$qualityBroadsheet \subseteq broadsheet$

$redTop \subseteq tabloid$

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

$whiteVanMan \subseteq \forall reads.tabloid$

$tabloid \cap broadsheet \subseteq \bot$

# 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

$driver \subseteq adult$

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

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

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

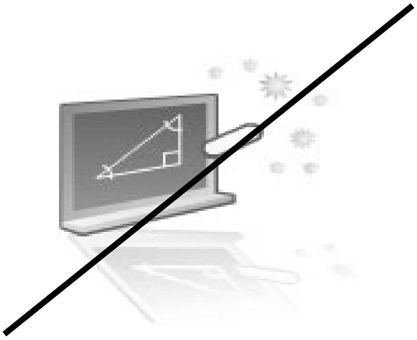
$haulageTruckDriver \equiv person \cap \exists drives.truck \cap$
$\quad \exists worksFor.(\exists partOf.haulageCompany)$

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

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

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

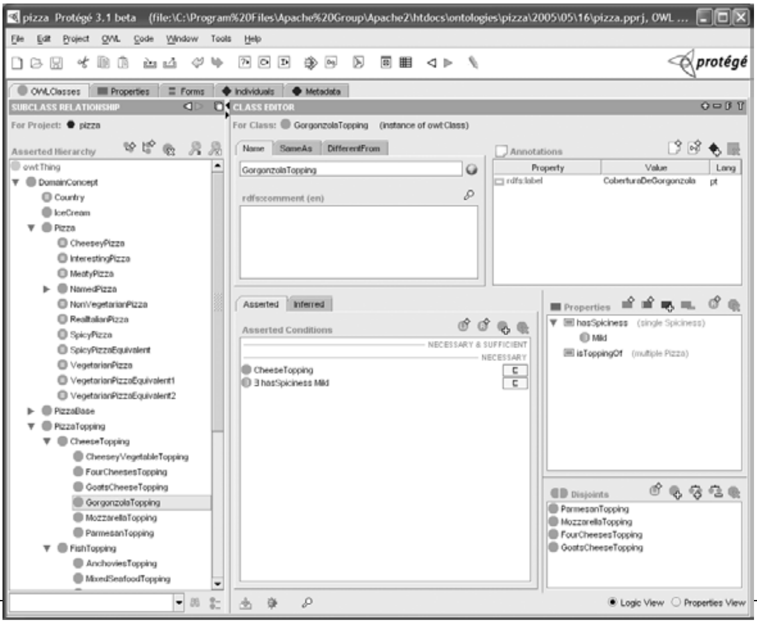
# OWL Example



# Protégé and the Protégé-OWL plug-in

- **http://protege.stanford.edu/**
- **Developed by Stanford Medical Informatics**
- **Features**
  - Software
    - Open source
    - Supports development of plugins to allow backend / interface extensions
  - Large user community (approx 30k)
  - Knowledge representation formalism
    - Core is based on Frames (object oriented) modelling
    - Open architecture that allows other modelling languages to be built on top (e.g., Protégé-OWL plug-in)

# Protégé-OWL



For Class: GorgonzolaTopping (instance of owl:Class)

GorgonzolaTopping

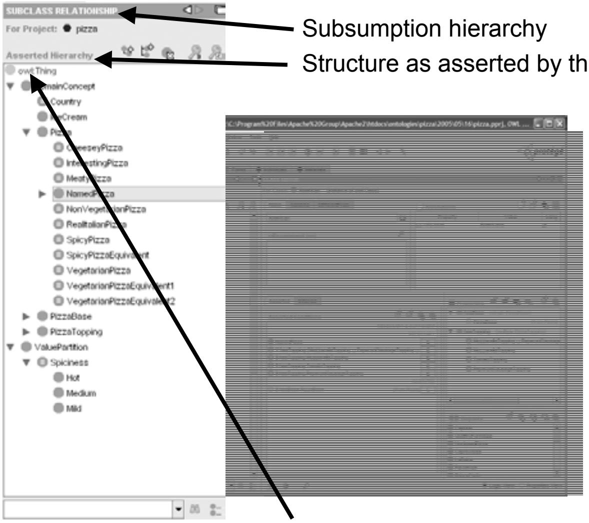
| Property | Value | Lang |
|---|---|---|
| rdfs:label | CoberturaDeGorgonzola | pt |

CheeseTopping ⊏
∃ hasSpiciness Mild ⊏

hasSpiciness (single Spiciness)
Mild
isToppingOf (multiple Pizza)

ParmesanTopping
MozzarellaTopping
FourCheesesTopping
GoatsCheeseTopping

# Class Hierarchy

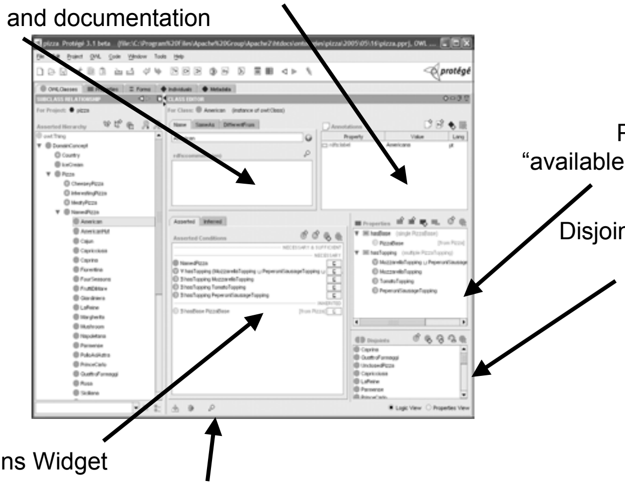


Subsumption hierarchy

Structure as asserted by the ontology engineer

owl:Thing is the root class

## Class Editor



Class annotations (for class metadata)

Class name and documentation

Properties “available” to Class

Disjoints widget

Conditions Widget

Class-specific tools (find usage etc)

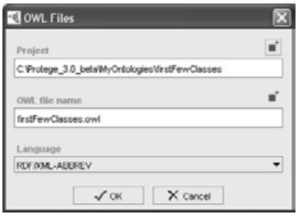
## Saving OWL Files

OWL = easy to make mistakes = save regularly

1. *Select File → Save Project As*
   *A dialog (as shown) will pop up*

2. *Select a file directly by clicking the button on the top right*
   *You will notice that 2 files are created*
   *.pprj – the project file*
   *this just stores information about the GUI and the workspace*
   *.owl – the OWL file*
   *this is where your ontology is stored in RDF/OWL format*

3. *Select OK*



OWL Files

Project: C:\Protege_3.0_beta\MyOntologies\firstFewClasses

OWL file name: firstFewClasses.owl

Language: RDF/XML-ABBREV

# Loading OWL files

1. *If you only have an OWL file:*

   ***- File→ New Project***

   *- Select **OWL Files** as the type*

   *- Tick **Create from existing sources***

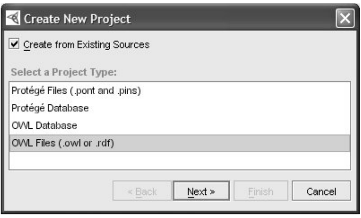
   *- **Next** to select the .owl file*

2. *If you've got a valid project file*:*

   ***- File → Open Project***

   *- select the .pprj file*

** ie one created on this version of Protégé - the s/w gets updated once every few days, so don't count on it unless you've created it recently– safest to build from the .owl file if in doubt*



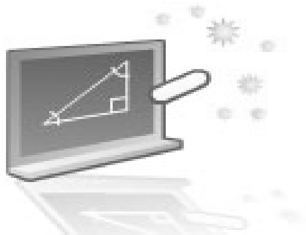
Create New Project

☑ Create from Existing Sources

Select a Project Type:

- Protégé Files (.pont and .pins)
- Protégé Database
- OWL Database
- OWL Files (.owl or .rdf)

# OWL Example

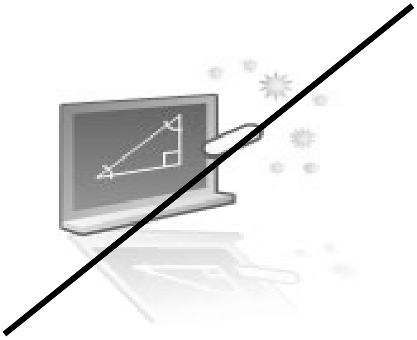


**Implement the previous sample ontology with the Protégé-OWL plug-in**

**- Practice with Protégé and the Protégé-OWL plug-in**

**- Develop parts of the previous ontology in groups, so that the ontologies can be integrated later**

# 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), ¬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

| | | |
|---|---|---|
| $x \bullet \{C_1 \sqcap C_2, \ldots\}$ | $\rightarrow_{\sqcap}$ | $x \bullet \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$ |
| $x \bullet \{C_1 \sqcup C_2, \ldots\}$ | $\rightarrow_{\sqcup}$ | $x \bullet \{C_1 \sqcup C_2, C, \ldots\}$<br>for $C \in \{C_1, C_2\}$ |
| $x \bullet \{\exists R.C, \ldots\}$ | $\rightarrow_{\exists}$ | $\begin{tikzcd} x \bullet \{\exists R.C, \ldots\} \arrow[d, "R"'] \\ y \bullet \{C\} \end{tikzcd}$ |
| $\begin{tikzcd} x \bullet \{\forall R.C, \ldots\} \arrow[d, "R"'] \\ y \bullet \{\ldots\} \end{tikzcd}$ | $\rightarrow_{\forall}$ | $\begin{tikzcd} x \bullet \{\forall R.C, \ldots\} \arrow[d, "R"'] \\ y \bullet \{C, \ldots\} \end{tikzcd}$ |
| $\begin{tikzcd} x \bullet \{\forall R.C, \ldots\} \arrow[d, "R"'] \\ y \bullet \{\ldots\} \end{tikzcd}$ | $\rightarrow_{\forall_+}$ | $\begin{tikzcd} x \bullet \{\forall R.C, \ldots\} \arrow[d, "R"'] \\ y \bullet \{\forall R.C, \ldots\} \end{tikzcd}$ |

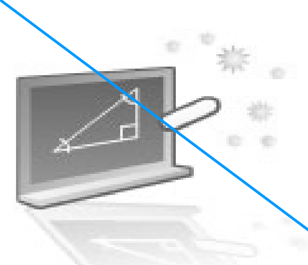
# Tableaux examples and exercises

- **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**
  - $\exists R.(C \vee D) \wedge \forall R.\neg C \wedge \neg \exists R.D$

# 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**

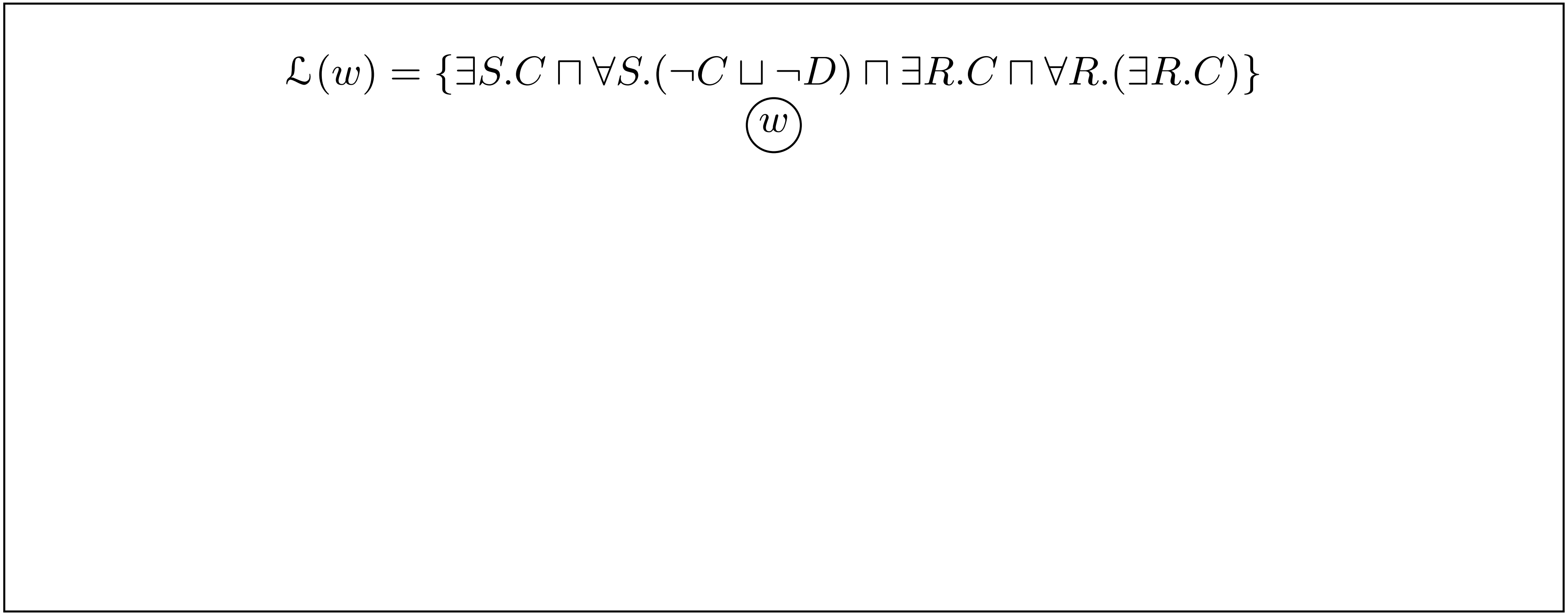
**Consistency checking: unsatisfiable restrictions in a Tbox (are the classes coherent?)**

# 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

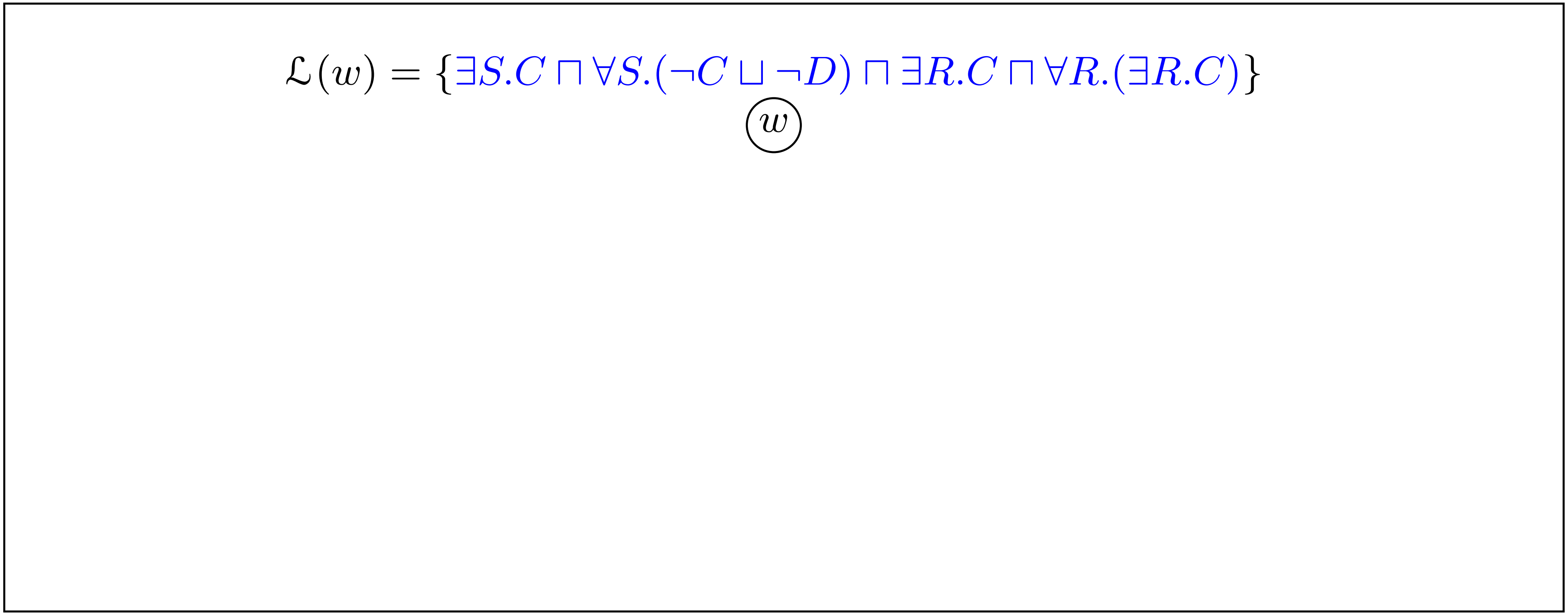
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

# Tableaux Algorithm — Example

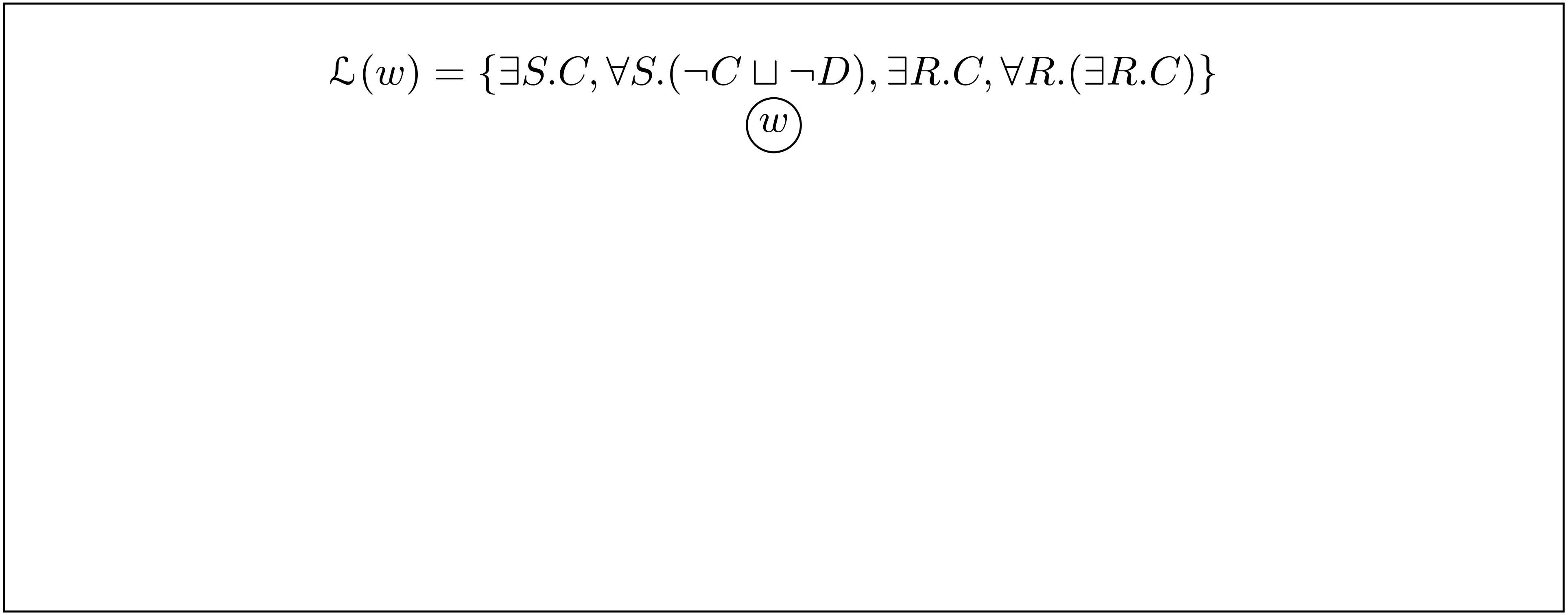
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

# Tableaux Algorithm — Example

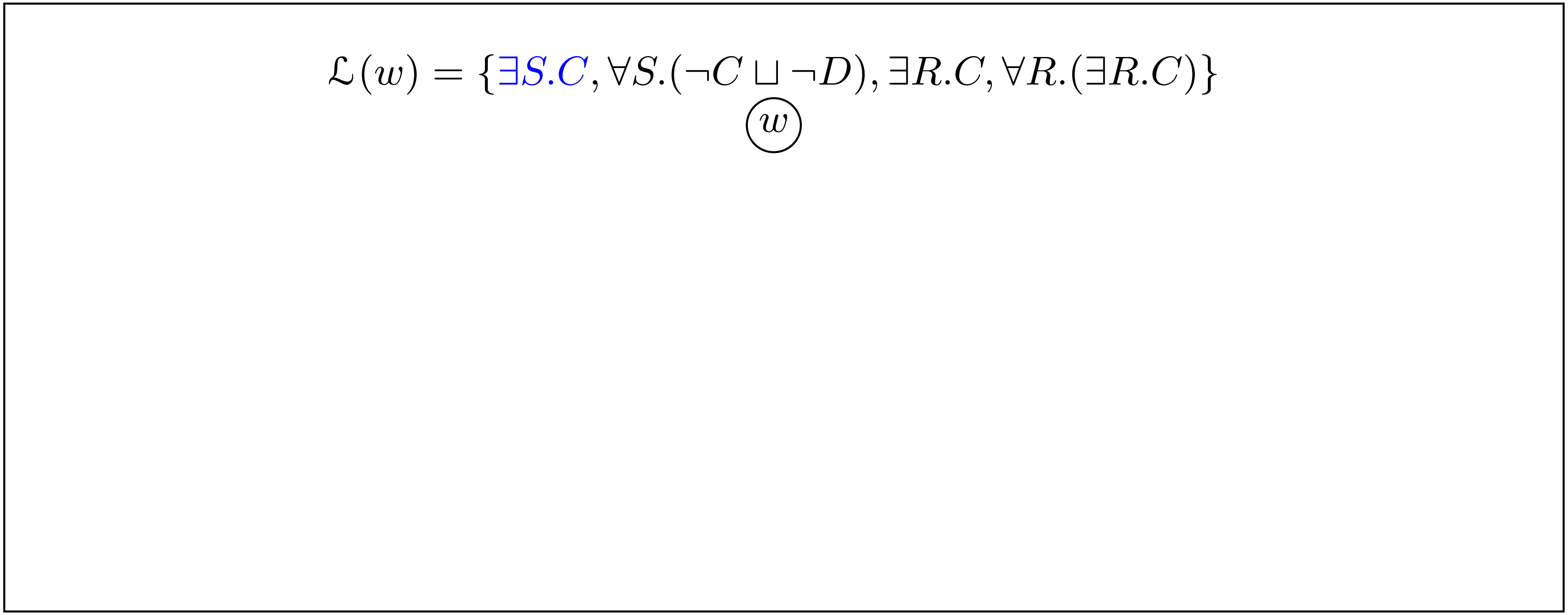
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

# Tableaux Algorithm — Example

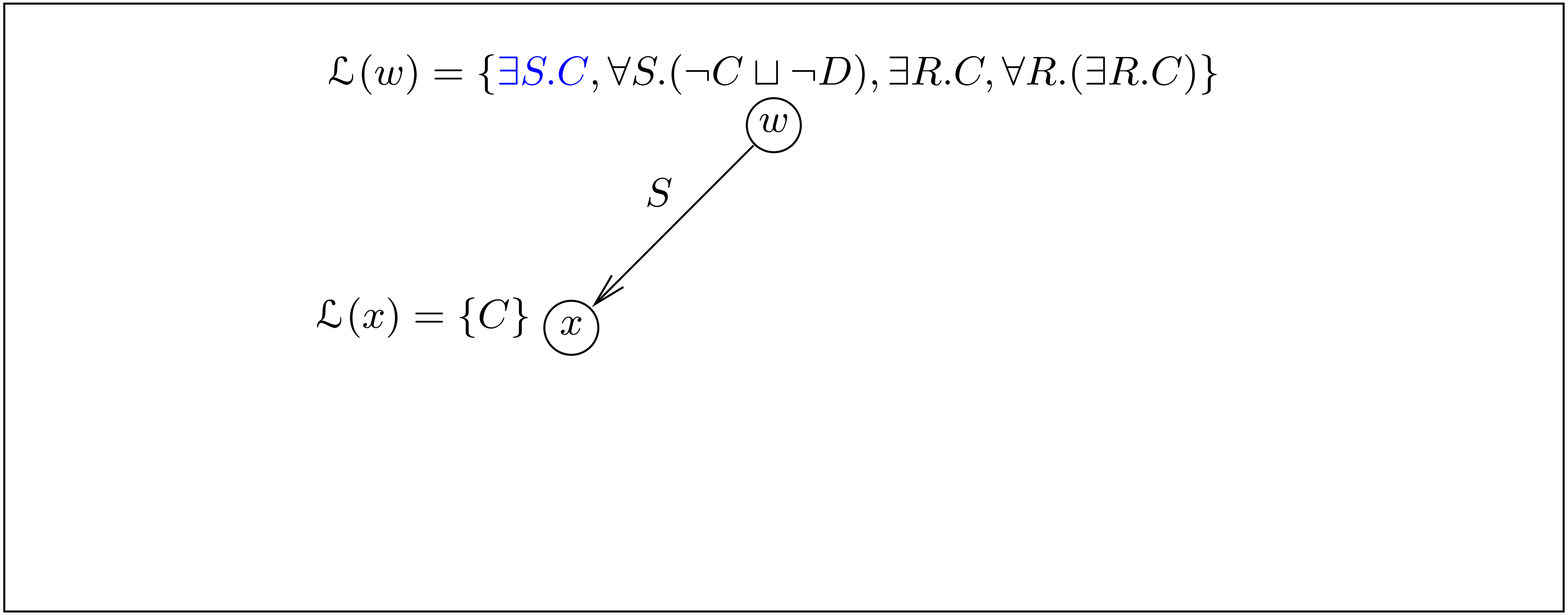
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

# Tableaux Algorithm — Example

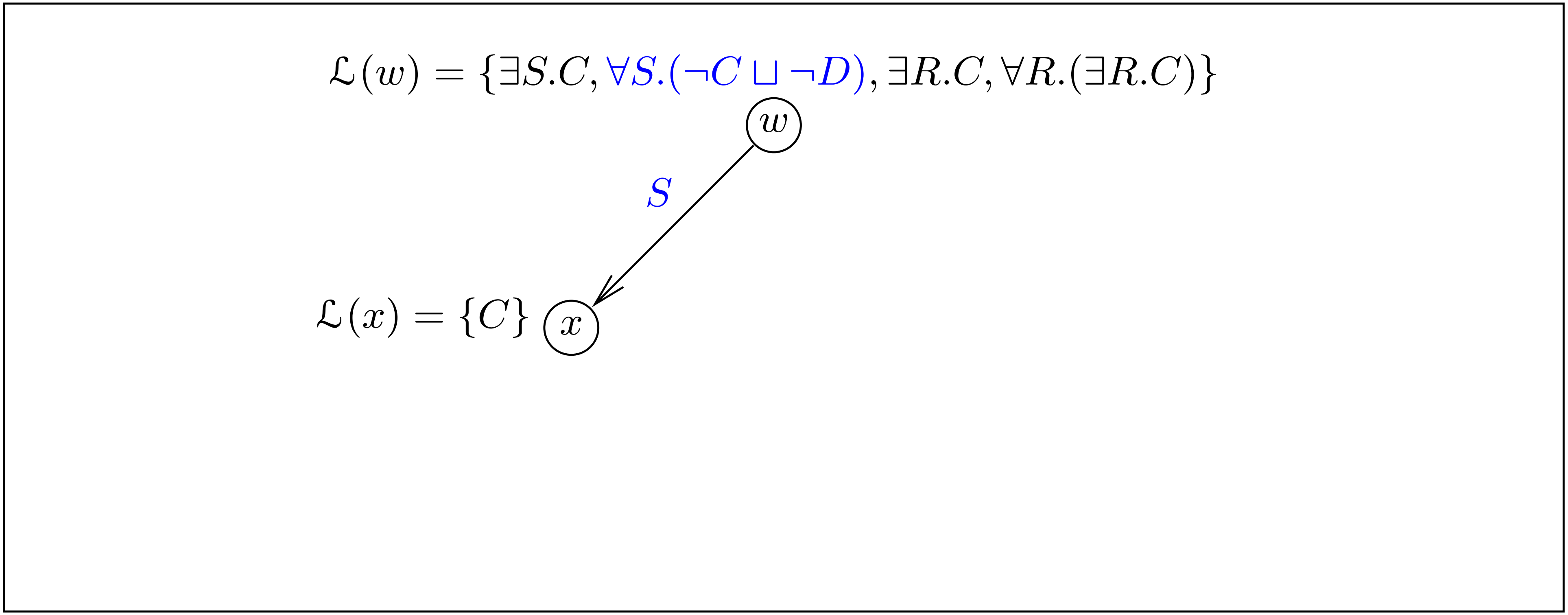
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

# 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

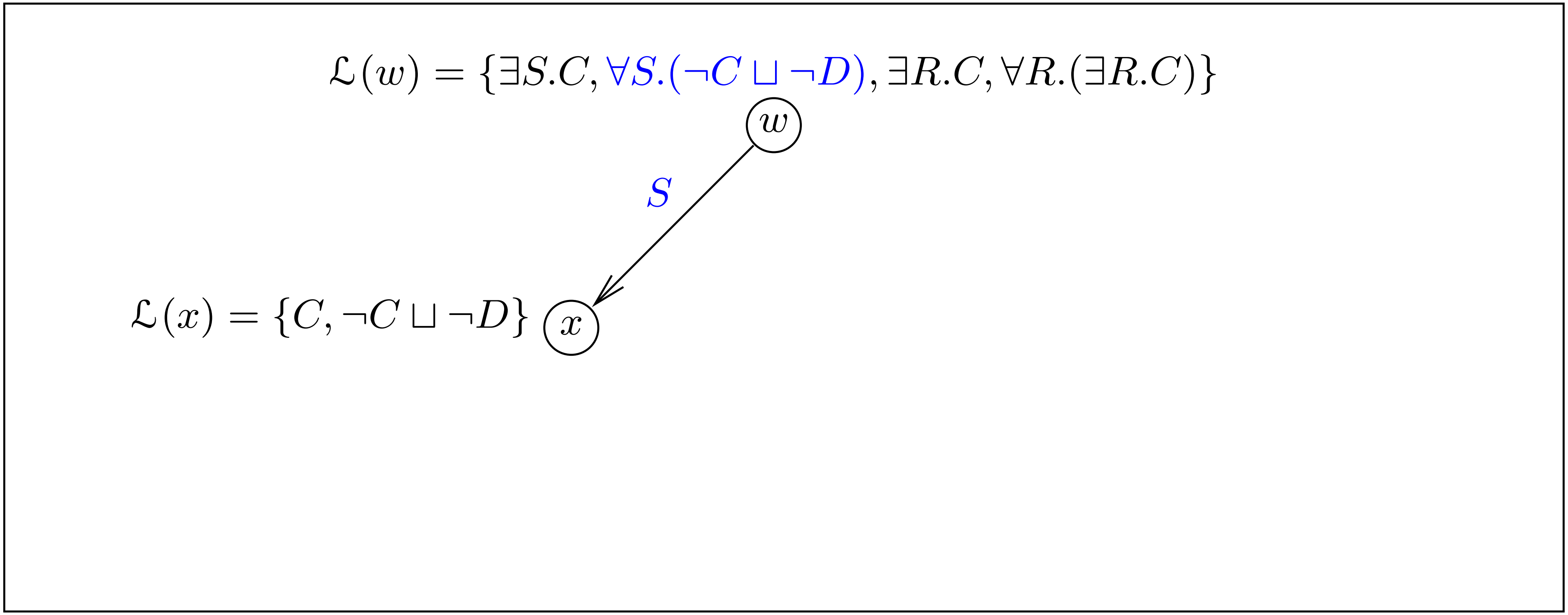


$$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$$

$$\mathcal{L}(x) = \{C\}$$

# 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

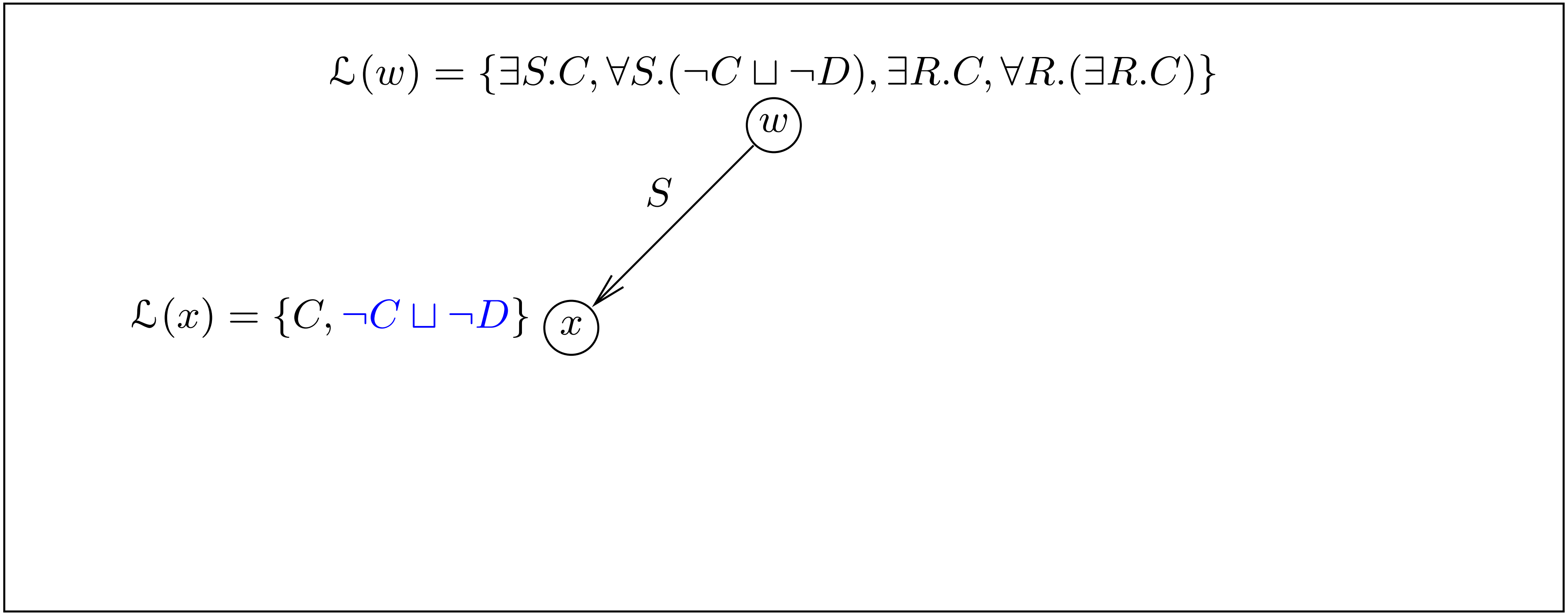


$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$

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

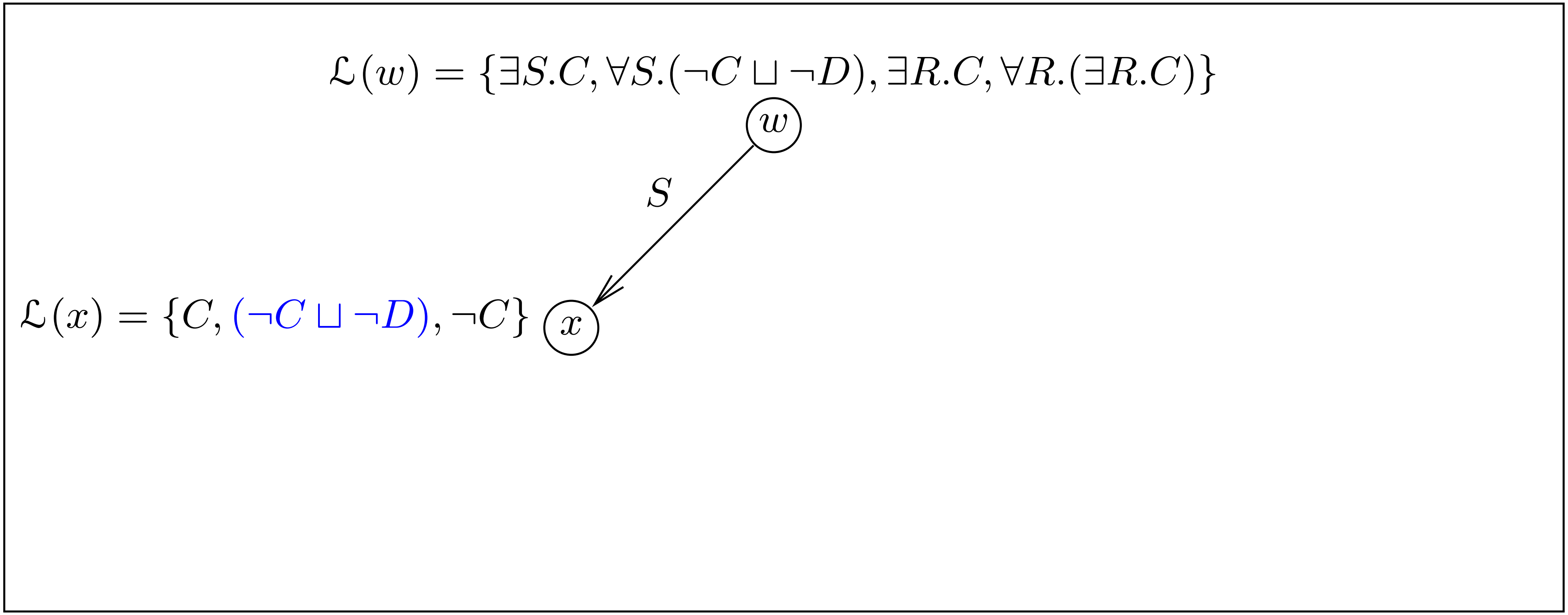
# Tableaux Algorithm — Example

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


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

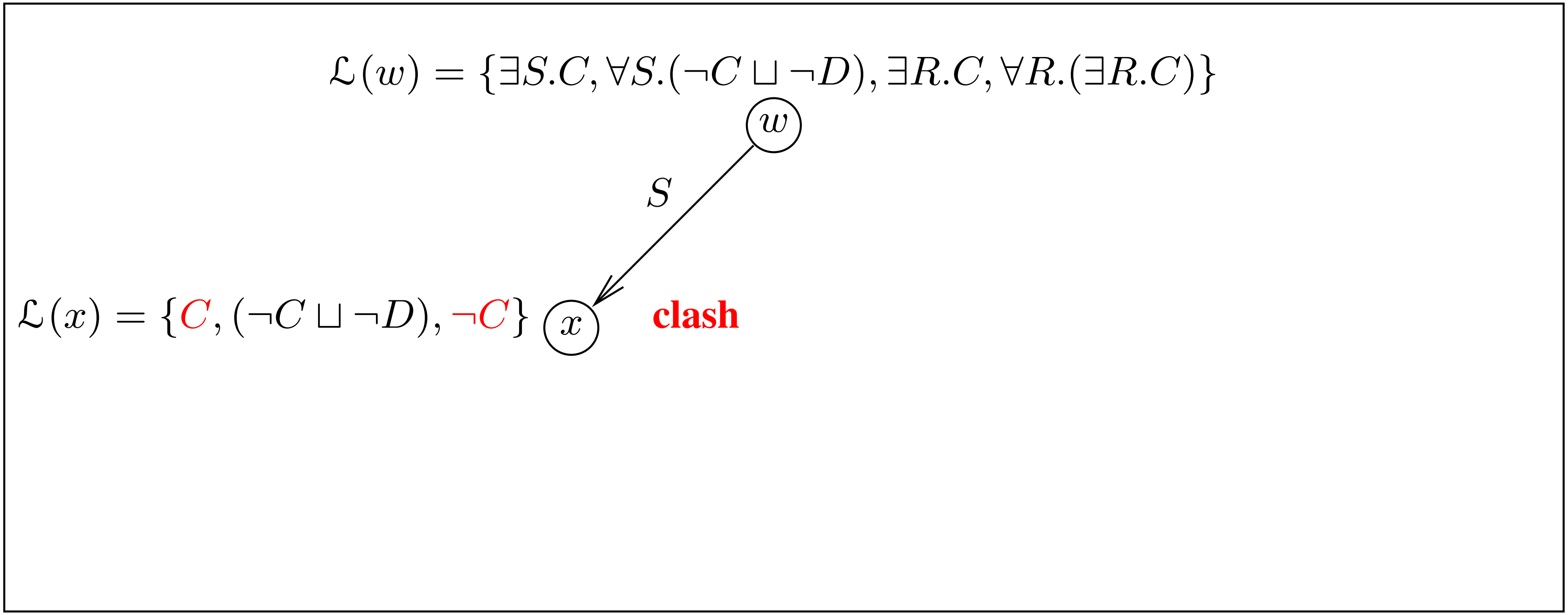
# Tableaux Algorithm — Example

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


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

# 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



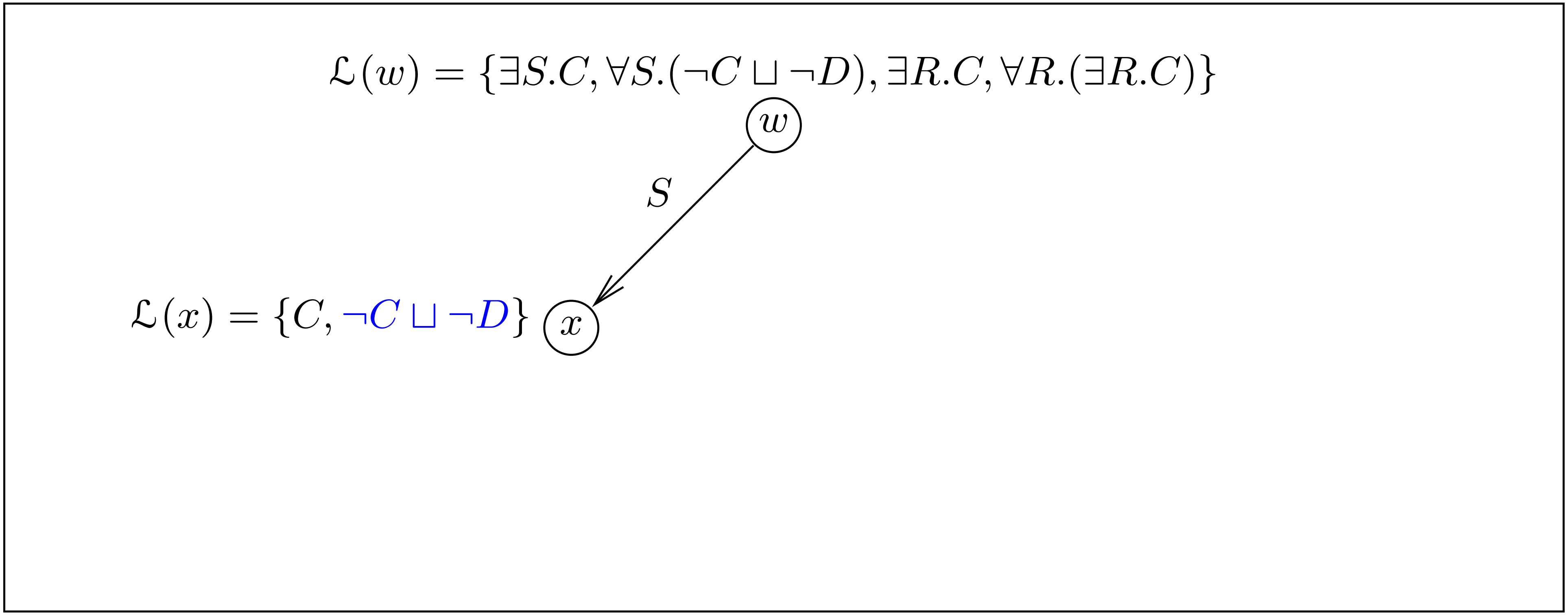
$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$

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

**clash**

# 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

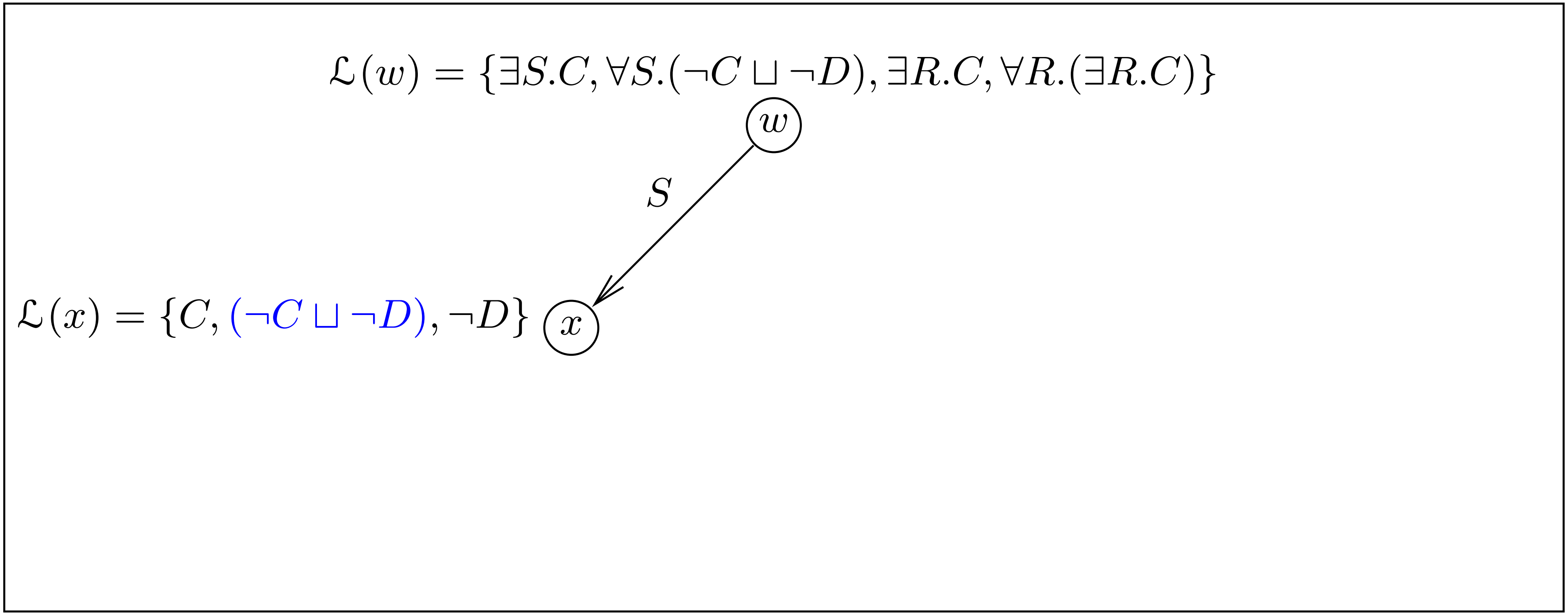


$\mathfrak{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$

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

# 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

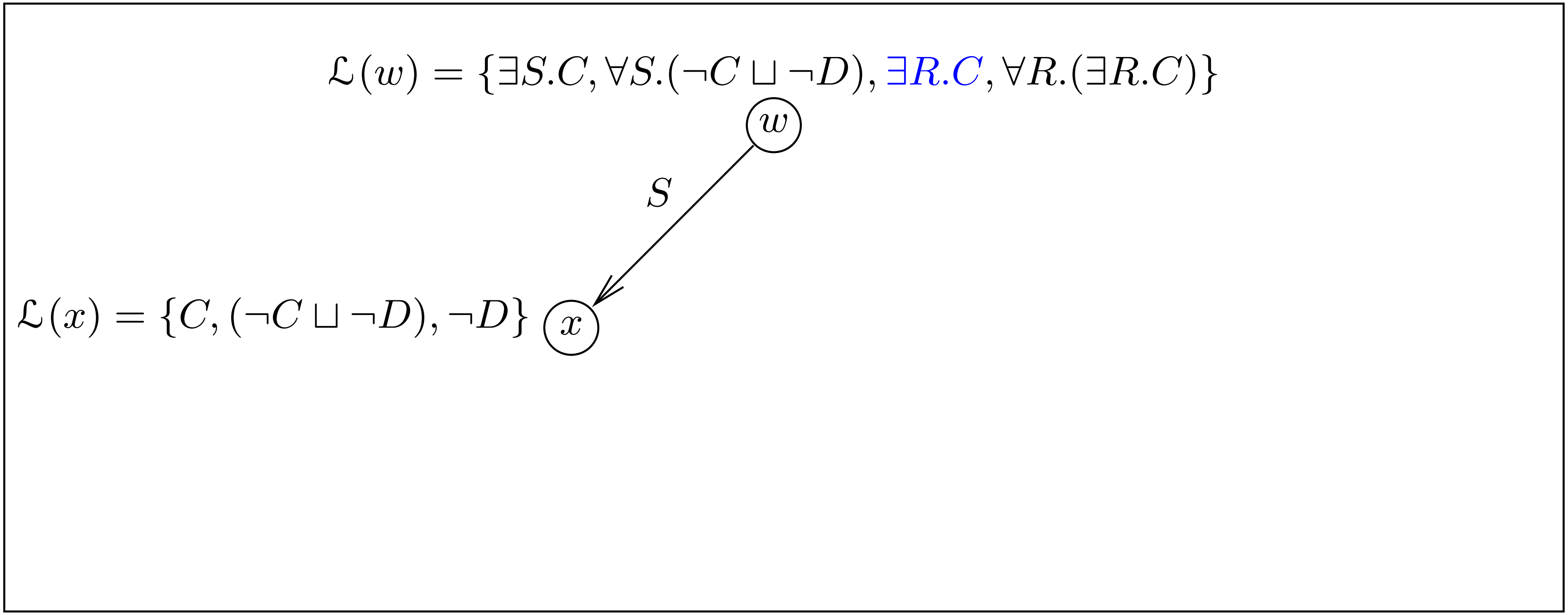


$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$

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

# 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

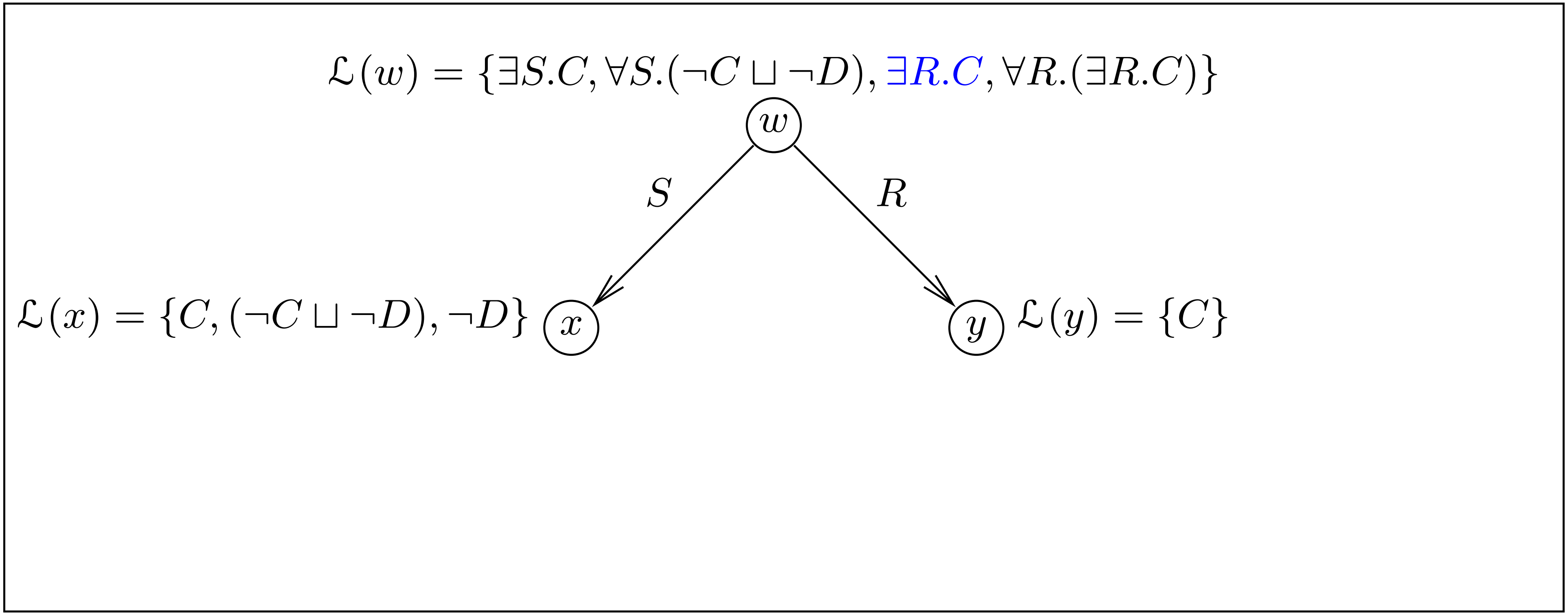


$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$

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

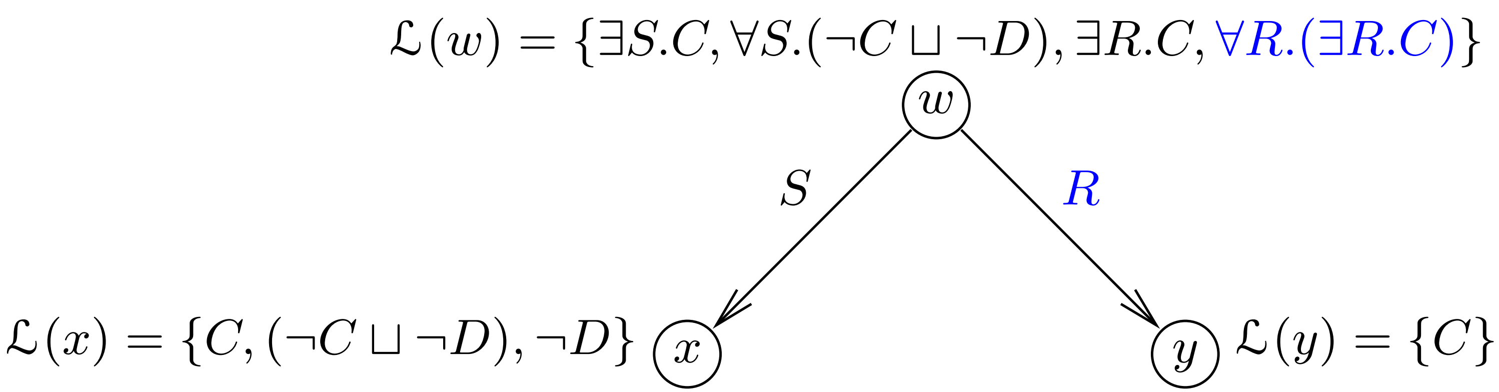
# Tableaux Algorithm — Example

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


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

# Tableaux Algorithm — Example

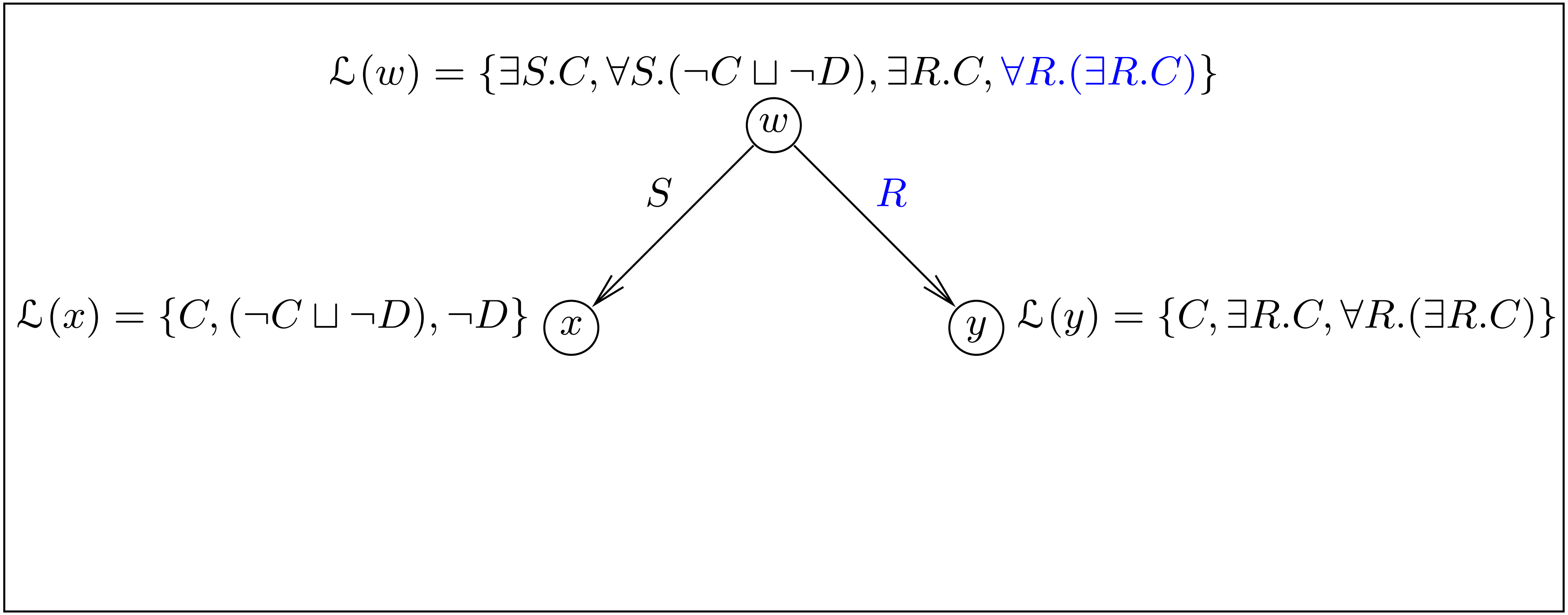
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

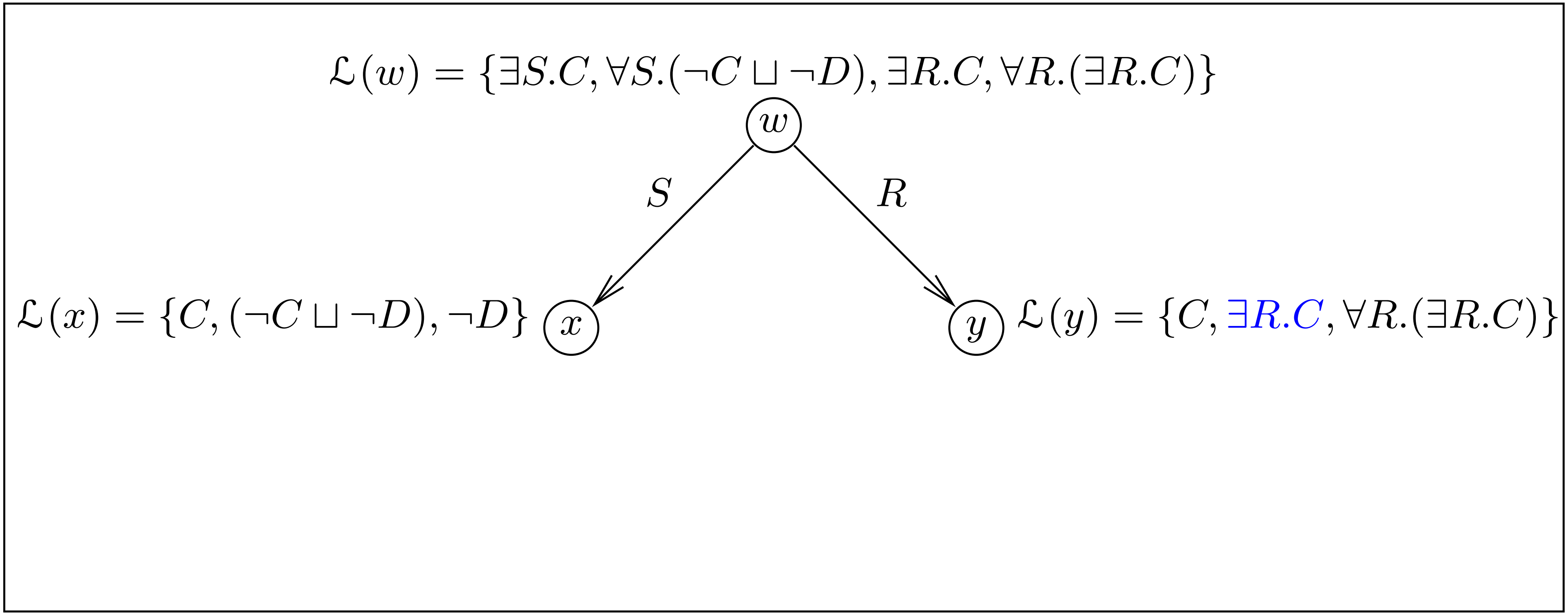
# Tableaux Algorithm — Example

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


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

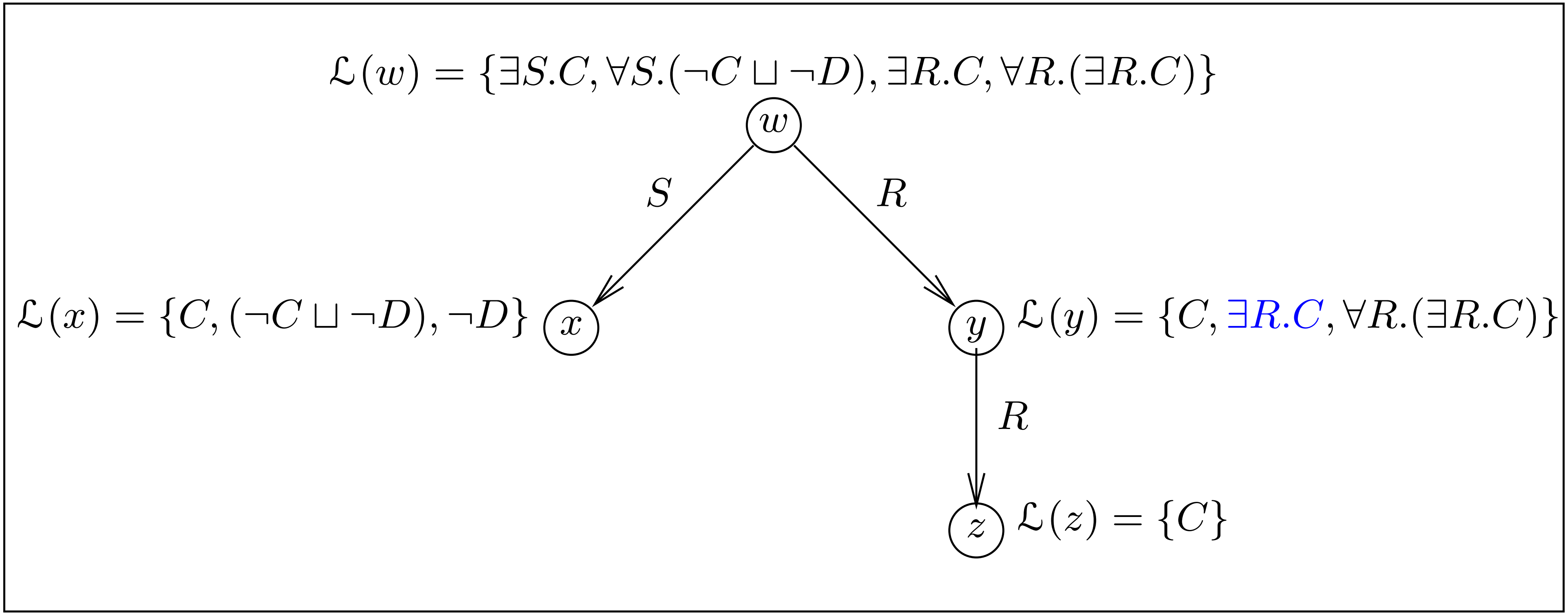
# Tableaux Algorithm — Example

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


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

# Tableaux Algorithm — Example

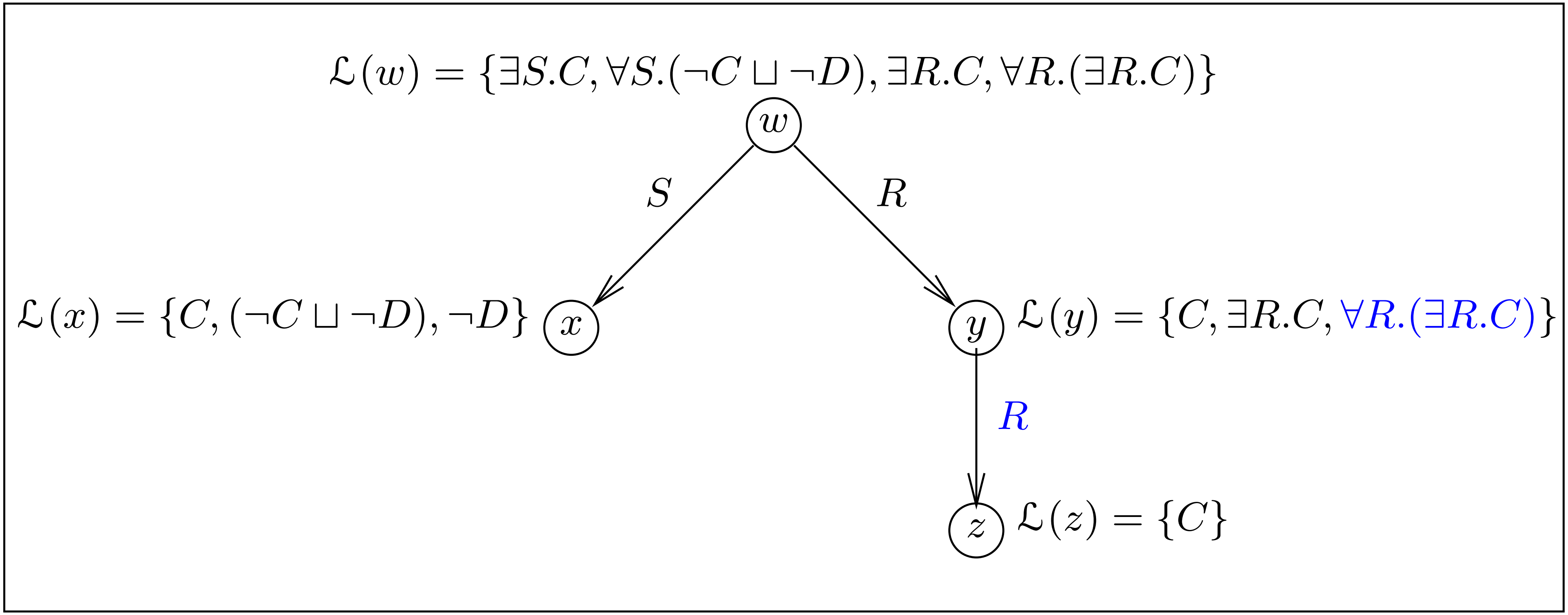
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role



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

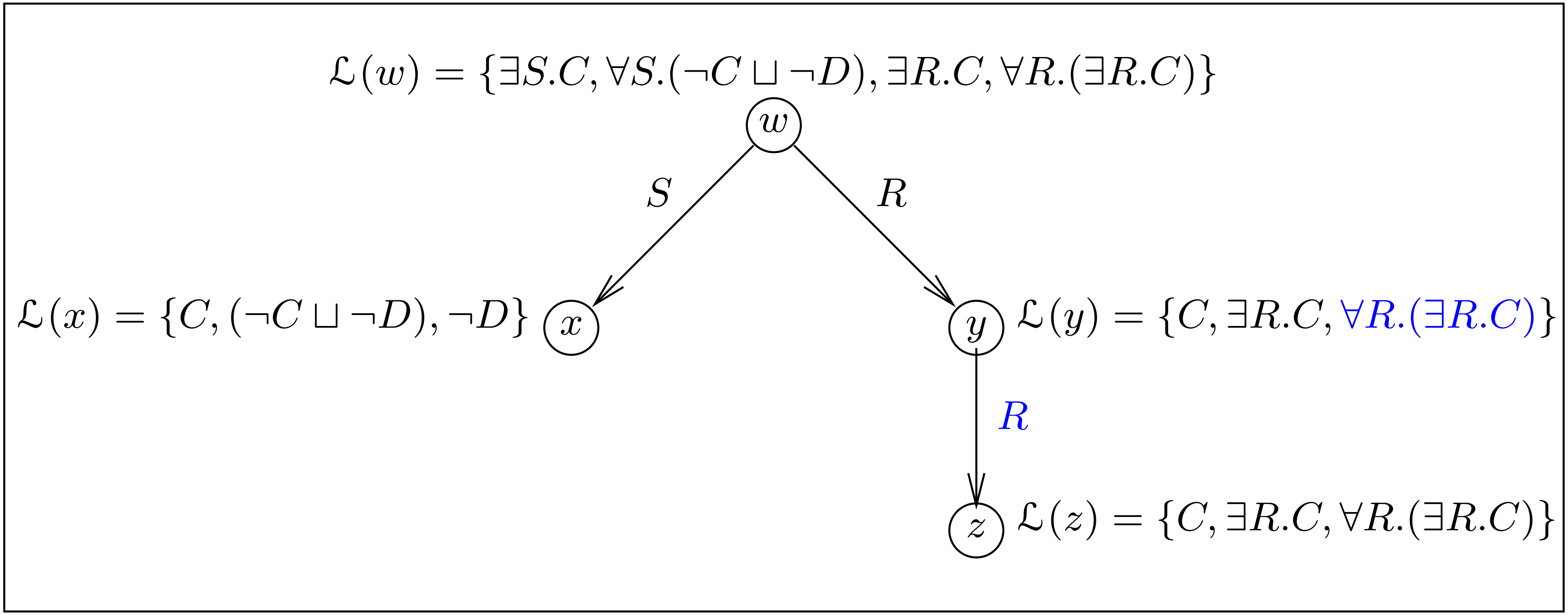
# Tableaux Algorithm — Example

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


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

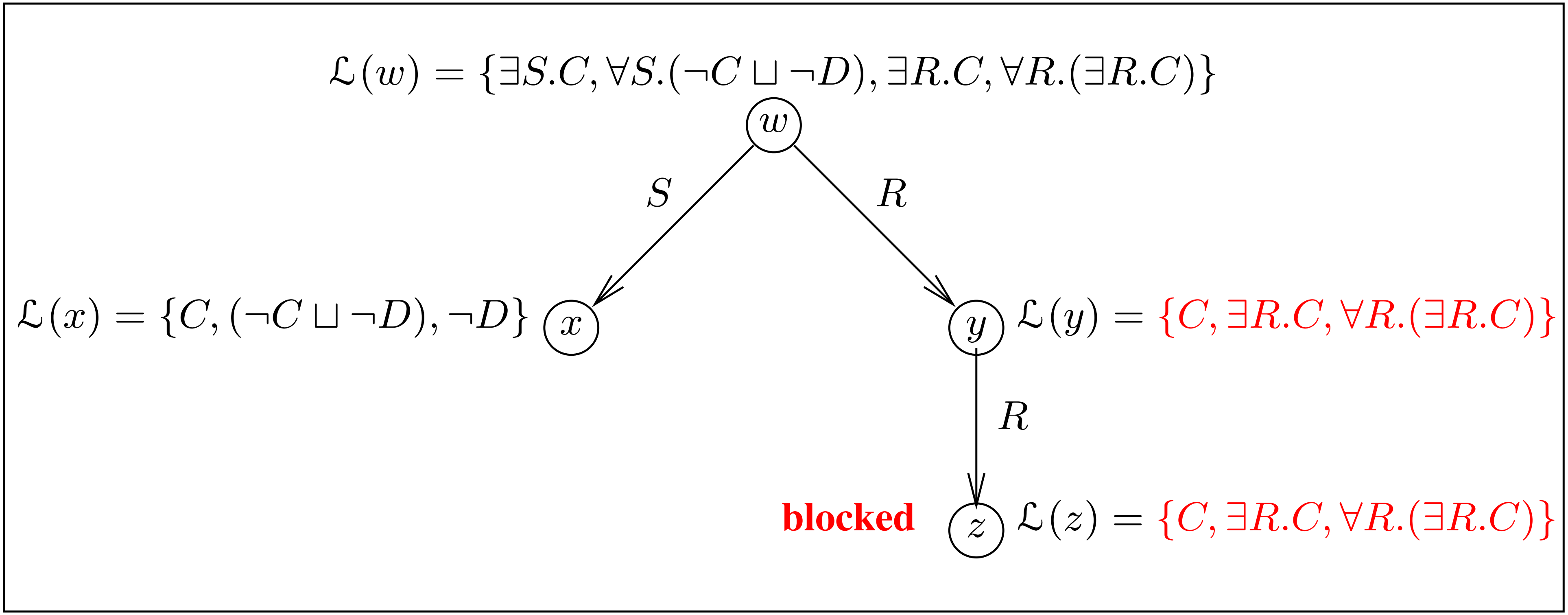
# Tableaux Algorithm — Example

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


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

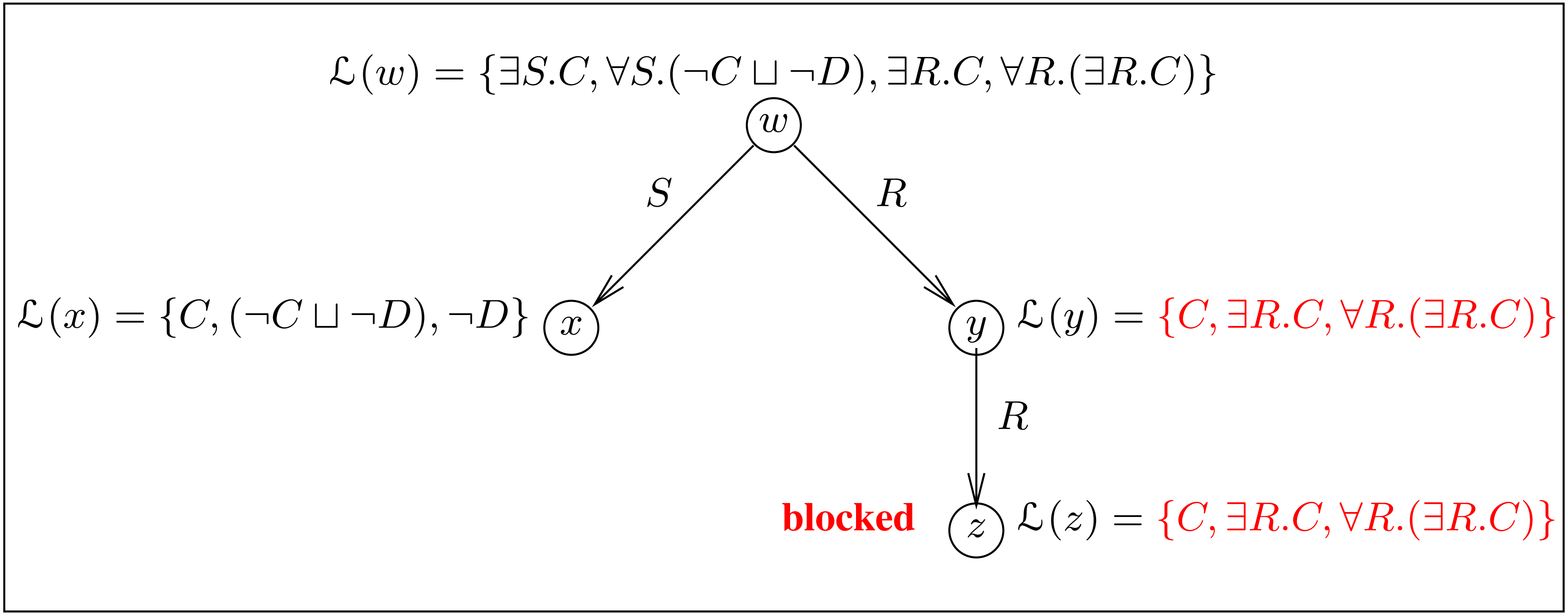
# Tableaux Algorithm — Example

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


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

# Tableaux Algorithm — Example

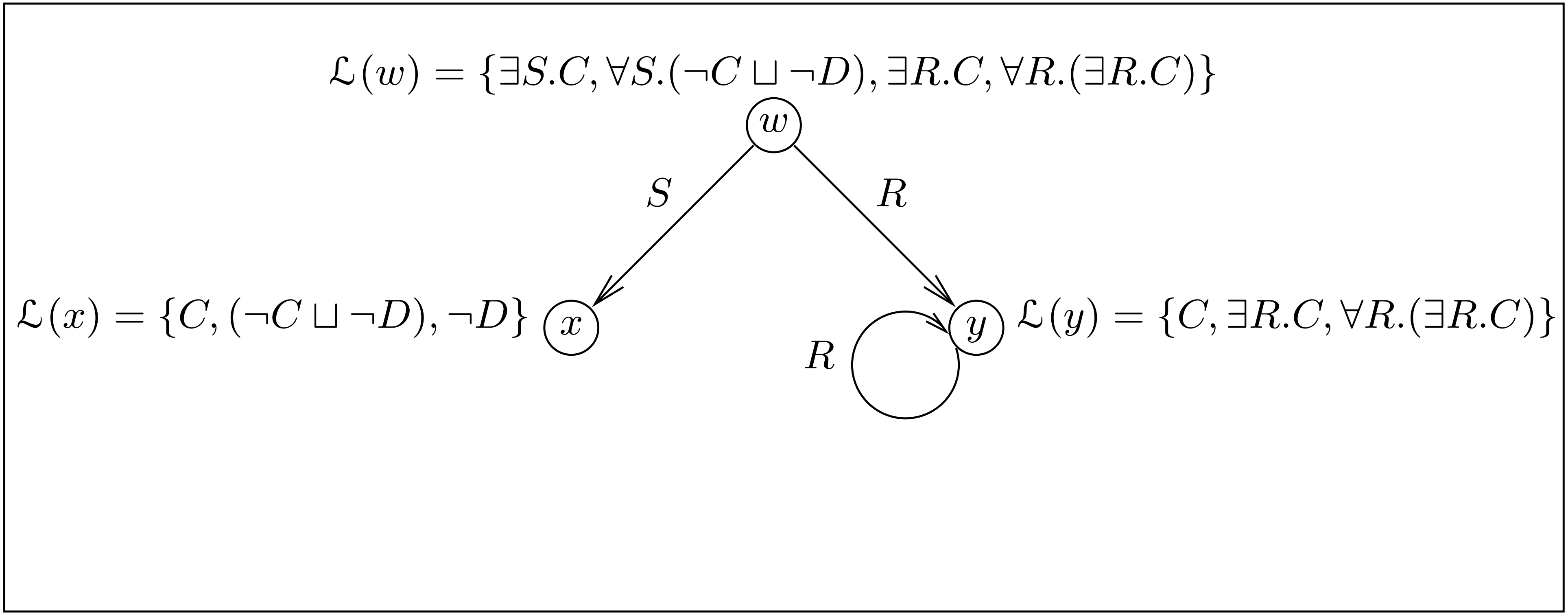
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where $R$ is a **transitive** role


$\mathfrak{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$
$w$
$S$
$R$
$\mathfrak{L}(x) = \{C, (\neg C \sqcup \neg D), \neg D\}$
$x$
$y$
$\mathfrak{L}(y) = \{C, \exists R.C, \forall R.(\exists R.C)\}$
$R$
blocked
$z$
$\mathfrak{L}(z) = \{C, \exists R.C, \forall R.(\exists R.C)\}$

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

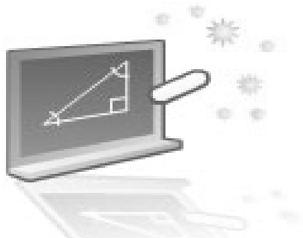

$\mathcal{L}(w) = \{\exists S.C, \forall S.(\neg C \sqcup \neg D), \exists R.C, \forall R.(\exists R.C)\}$
$w$
$S$
$R$
$\mathcal{L}(x) = \{C, (\neg C \sqcup \neg D), \neg D\}$
$x$
$R$
$y$
$\mathcal{L}(y) = \{C, \exists R.C, \forall R.(\exists R.C)\}$

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

# 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**
  - $\exists R.(C \vee D) \wedge \forall R.\neg C \wedge \neg\exists R.D$

# 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.**

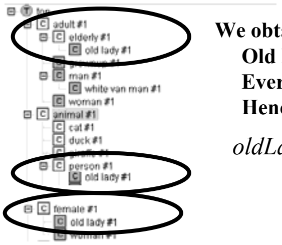
$$elderly \subseteq person \cap adult$$

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

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

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

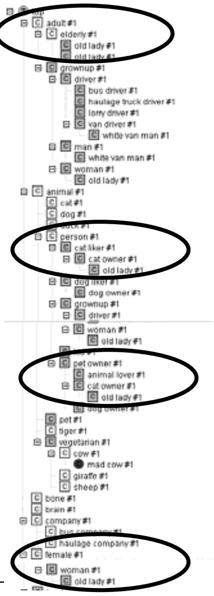
$$oldLady \subseteq \exists hasPet.animal \cap \forall hasPet.cat$$



```
top
  adult #1
    elderly #1
      old lady #1
    grownup #1
    man #1
      white van man #1
    woman #1
  animal #1
    cat #1
    duck #1
    giraffe #1
    person #1
      old lady #1
  female #1
    old lady #1
    woman #1
```

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

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



```
top
  adult #1
    elderly #1
      old lady #1
      old lady #1
    grownup #1
      driver #1
        bus driver #1
        haulage truck driver #1
        lorry driver #1
        van driver #1
          white van man #1
      man #1
        white van man #1
      woman #1
        old lady #1
  animal #1
    cat #1
    dog #1
    duck #1
    person #1
      cat liker #1
        cat owner #1
          old lady #1
      dog liker #1
        dog owner #1
      grownup #1
        driver #1
        woman #1
          old lady #1
      pet owner #1
        animal lover #1
        cat owner #1
          old lady #1
        dog owner #1
    pet #1
    tiger #1
    vegetarian #1
      cow #1
        mad cow #1
      giraffe #1
      sheep #1
  bone #1
  brain #1
  company #1
    bus company #1
    haulage company #1
  female #1
    woman #1
      old lady #1
```

## 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

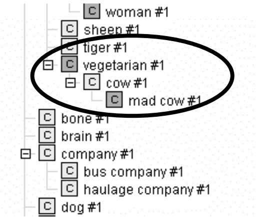
**Cows are vegetarian.**
**A vegetarian is an animal that does not eat animals nor parts of animals.**
**A mad cow is a cow that eats brains that can be part of a sheep**

$$cow \subseteq vegetarian$$

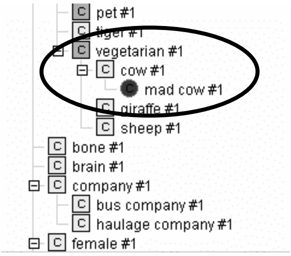
$$vegetarian \equiv animal \cap \forall eats.\neg animal \cap$$
$$\forall eats.\neg(\exists partOf.animal))$$

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

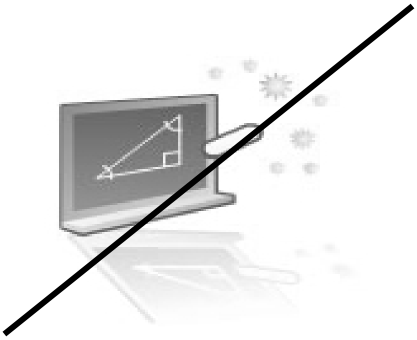
$$(animal \cup \exists partOf.animal) \cap (plant \cup \exists partOf.plant) \subseteq \perp$$



**We obtain:**
**Mad cow is unsatisfiable**



## OWL Example

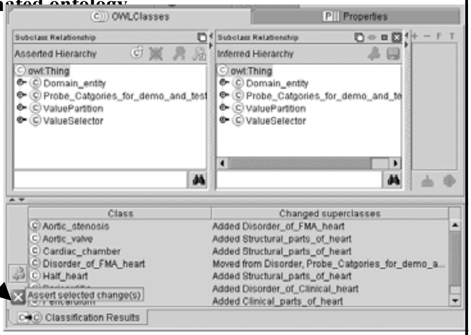


# When to use a classifier

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**



Asserted Hierarchy:
- owl:Thing
  - Domain_entity
  - Probe_Catgories_for_demo_and_test
  - ValuePartition
  - ValueSelector

Inferred Hierarchy:
- owl:Thing
  - Domain_entity
  - Probe_Catgories_for_demo_and_te
  - ValuePartition
  - ValueSelector

| Class | Changed superclasses |
|---|---|
| Aortic_stenosis | Added Disorder_of_FMA_heart |
| Aortic_valve | Added Structural_parts_of_heart |
| Cardiac_chamber | Added Structural_parts_of_heart |
| Disorder_of_FMA_heart | Moved from Disorder, Probe_Catgories_for_demo_a... |
| Half_heart | Added Structural_parts_of_heart |
| Pericarditis | Added Disorder_of_Clinical_heart |
| Pericardium | Added Clinical_parts_of_heart |

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

gene

protein

cell

expression

Lung

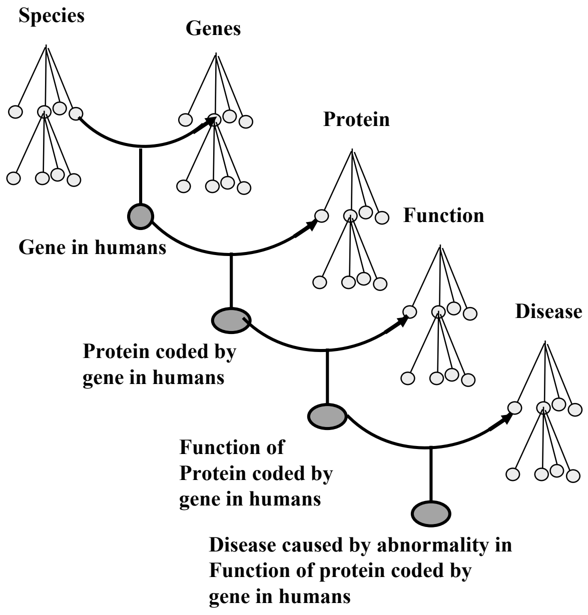
inflammation

infection

bacterial

deletion

polymorphism



Species

Genes

Protein

Function

Disease

Gene in humans

Protein coded by gene in humans

Function of Protein coded by gene in humans

Disease caused by abnormality in Function of protein coded by gene in humans

**Logical Constructs build complex concepts from modularised primitives**

# Rationale for Normalisation/Modularisation

- **Normalisation in order to make large ontologies easier to maintain**
  - Trees (classes have only one parent) are easier to maintain
  - Directed Acyclic Graphs (classes can have several parents) can be derived with a reasoner from the trees

- **Objective: derive explicit distinctions between modules**
  - Primitives are opaque to the reasoner
    - Information implicit in primitive names cannot contribute to modularisation
  - Primitives are indivisible to both human and reasoner
    - Each primitive should represent a single notion
  - Therefore, each primitive must belong to exactly one module
    - If a primitive belongs to two modules, they are not modular.
    - If a primitive belongs to two modules, it probably conflates two notions
  - Therefore concentrate on the "primitive skeleton" of the domain ontology

# Normalisation Criteria (I)

1. **The skeleton should consist of disjoint trees**
   - Every primitive concept should have exactly one primitive parent
     - All multiple hierarchies are the result of inference by reasoner

2. **No hidden changes in meaning**
   - Each branch should be homogeneous and logical ("Aristotelian")
     - Hierarchical principle should be subsumption
       - Otherwise we are "lying to the logic"
     - The criteria for differentiation should follow consistent principles in each branch
   - Example of non-homogeneous taxonomy (from The Celestial Emporium of Benevolent Knowledge, Borges)
     - "On those remote pages it is written that animals are divided into:
       - a. those that belong to the Emperor b. embalmed ones
       - c. those that are trained d. suckling pigs
       - e. mermaids f. fabulous ones
       - g. stray dogs h. those that are included in this classification
       - i. those that tremble as if they were mad j. innumerable ones
       - k. those drawn with a very fine camel's hair brush l. others
       - m. those that have just broken a flower vase n. those that resemble flies from a distance"

# Normalisation Criteria (II)

## 3. Distinguish "Self-standing" and "Refining" Concepts

- Self-standing concepts (person, idea, plant, committee, belief, etc.)
  - Roughly Welty & Guarino's "sortals"
- Refining concepts – depend on self-standing concepts (mild|moderate|severe, hot|cold, etc.)
  - Roughly Welty & Guarino's non-sortals, Smith's "fiat partitions", Value Types for engineers

## 3a. Self-standing primitives should be globally disjoint & open

- Primitives are disjoint
  - If primitives overlap, the overlap conceals implicit information
- A list of self-standing primitives can never be guaranteed complete
  - How many kinds of person? of plant? of committee? of belief?
  - Can't infer: Parent & ¬sub1 &…& ¬subn-1 → subn

## 3b. Refining primitives should be locally disjoint & closed

- Individual values must be disjoint, but can be hierarchical
  - e.g. "very hot", "moderately severe"
- Each list can be guaranteed to be complete
- "Value partitions" themselves need not be disjoint
  - "being hot" is not disjoint from "being severe"
    - Allowing Valuetypes to overlap is a useful trick, e.g. restriction has_state someValuesFrom (severe and hot)

# Normalisation Criteria (III)

## 4. Axioms

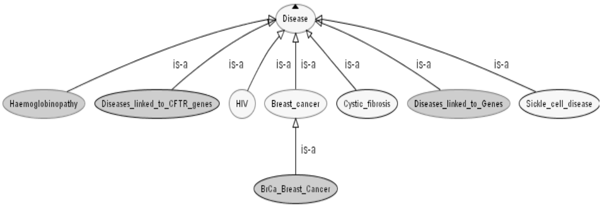
- No axiom should denormalise the ontology
- No axiom should imply that a primitive is part of more than one branch of primitive skeleton
  - If all primitives are disjoint, any such axioms will make that primitive unsatisfiable
  - A partial test for normalisation:
    - Create random conjunctions of primitives which do not subsume each other.
    - If any are satisfiable, the ontology is not normalised

# Consequences

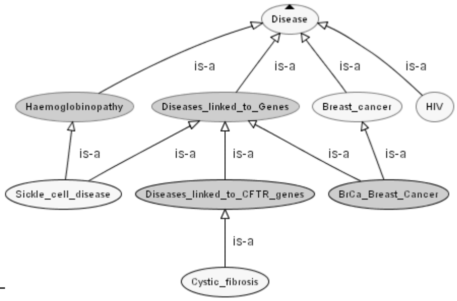
- **All self-standing primitives are disjoint**
- **All multiple classification is inferred**
- **For any two primitive self-standing classes, either one subsumes the other or they are disjoint**
- **Every self standing concept is part of exactly one primitive branch of the skeleton**
- **Every self-standing concept has exactly one most specific primitive ancestor**
- **Primitives introduced by a conjunction of one class and a boolean combination of zero or more restrictions**
  - Tree subclass-of Plant and
    restriction isMadeOf someValuesFrom Wood
  - Resort subclass-of Accommodation
    restriction isIntendedFor someValueFrom Holidays
- **Use of axioms limited (outside of skeleton construction). The following are a safe but not exhaustive guide:**
  - The right side of subclass axioms limited to restrictions
  - Both sides of disjointness axioms limited to restrictions
  - No equivalence axioms with primitives on either side

# Example (I)

- **Build a simple tree**

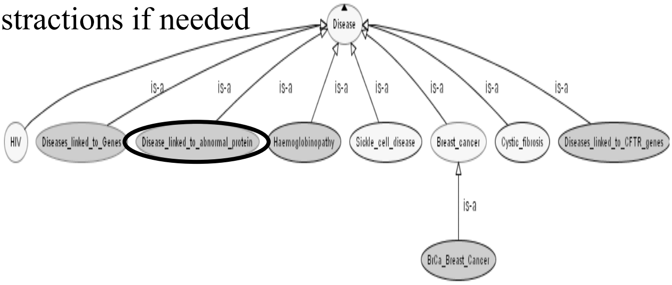


- **Let the classifier organise it and check consistency**

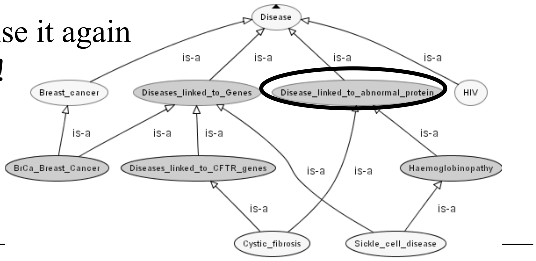


# Example (II)

- Add more abstractions if needed



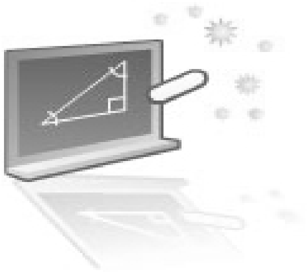
- Let the classifier organise it again and check consistency!!



# 3. Inference

- **We have already seen some examples**
  - And we will see some more in session 4

# Exercise



•Objective
- Apply design principles to normalise/modularise the OWL ontology developed in the previous exercises

•Tasks
- Identify disjoint trees in the ontology developed.
- Distinguish self-standing and refining classes.
- Identify axioms.
- Go back to the first task if the ontology is not normalised yet.

# 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

# How can we implement ontologies?
# Ontology languages

**Asunción Gómez-Pérez**
**Mariano Fernández-López**
**Oscar Corcho**
asun@fi.upm.es, mfernandez.eps@ceu.es, ocorcho@cs.man.ac.uk
Grupo de Ontologías
Laboratorio de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
Campus de Montegancedo sn,
28660 Boadilla del Monte, Madrid, Spain