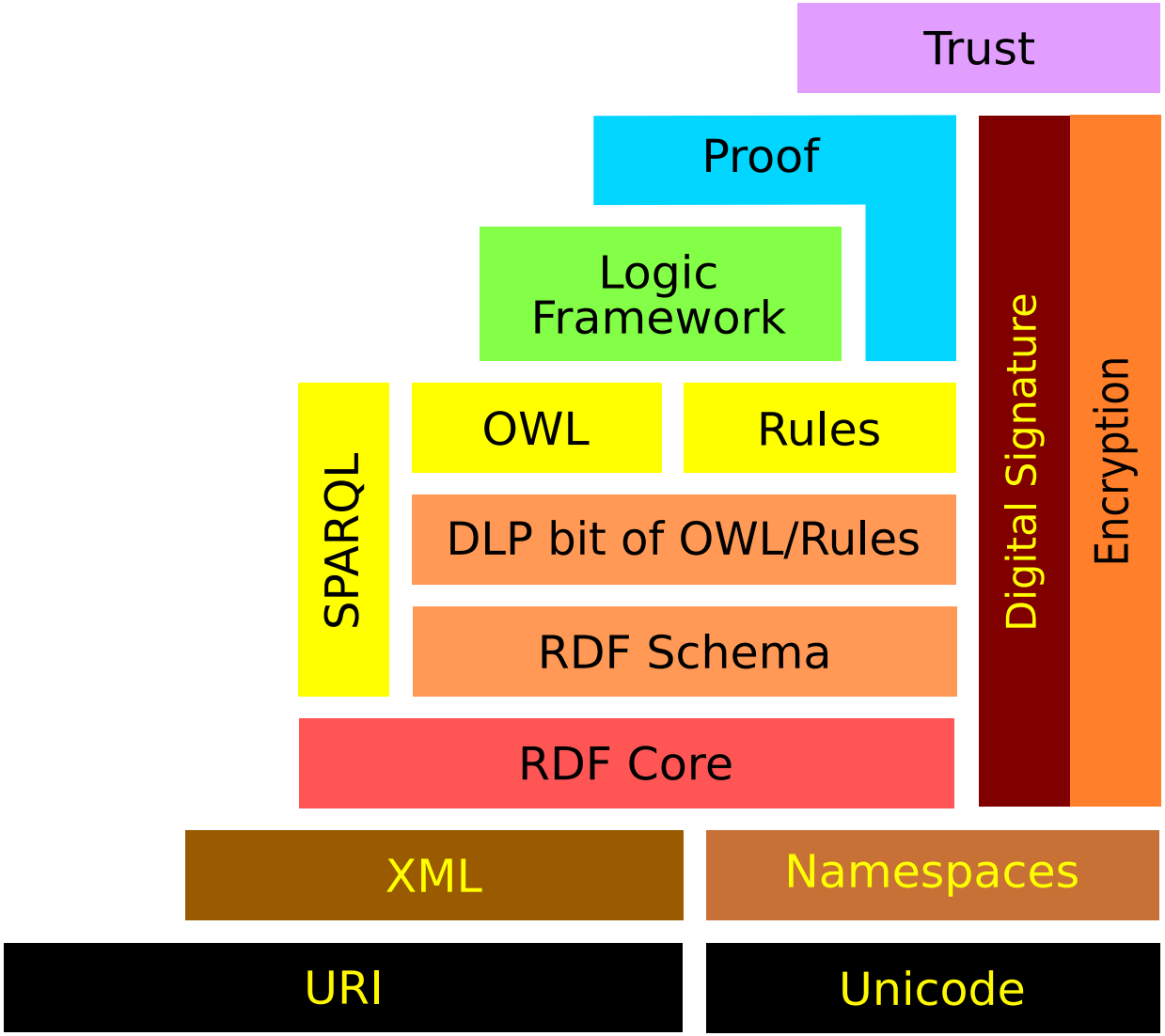
OWL 2: Description Logics, Inference

Jose Luis Ambite
University of Southern California

Semantic Web Layer Cake



slide by Pedro Szekely

Ontology

An ontology is a formal specification of a shared conceptualization [Gruber, 1992]

- An ontology defines (specifies) the concepts, relationships, and other distinctions that are relevant for modeling a domain.
- Represented in a formal language (logic), which provides semantics and inference mechanisms to ensure (facilitate) coherent use.

RDF Schema (RDFS)

- RDF is a universal language that lets users describe resources in their own vocabularies
  - However, RDF does not assume, nor does it define semantics of any particular application domain
- RDF Schema can define semantics of the data using:
  - Classes and Properties
  - Class Hierarchies and Inheritance
  - Property Hierarchies
  - Property Domain and Ranges
- RDF Schema: W3C recommendation, Feb 2004
  - http://www.w3.org/RDF/

Classes and their Instances

- We must distinguish between
  - Concrete "things" (*individuals*) in the domain: Craig, CSCI548, USC, etc.
  - Sets of individuals sharing some properties (*classes*): professors, courses, universities, etc.
- Individual objects that belong to a class are referred to as instances of that class
- The relationship between instances and classes in RDF is through rdf:type

Class Hierarchies

- Classes can be organized in hierarchies
  - A is a subclass of B if every instance of A is also an instance of B
  - In logic: $\forall x\ A(x) \rightarrow B(x)$
  - B is a superclass of A
- A subclass graph need not be a tree
- A class may have multiple superclasses

RDF Schema (Triples Notation)

A schema defines the terms that will be used in the RDF statements and gives specific meanings to them.

http://www.w3.org/TR/rdf-schema/

Example:

ex:MotorVehicle rdf:type rdfs:Class . → MotorVehicle is an instance of rdfs:Class
ex:PassengerVehicle rdf:type rdfs:Class .
ex:Van rdf:type rdfs:Class .
ex:Truck rdf:type rdfs:Class .
ex:MiniVan rdf:type rdfs:Class .

PassengerVehicle is a subclass of MotorVehicle

ex:PassengerVehicle rdfs:subClassOf ex:MotorVehicle .
ex:Van rdfs:subClassOf ex:MotorVehicle .
ex:Truck rdfs:subClassOf ex:MotorVehicle .
ex:MiniVan rdfs:subClassOf ex:Van .
ex:MiniVan rdfs:subClassOf ex:PassengerVehicle . → Multiple Inheritance

RDF Schema (RDF/XML notation)

```
<rdf:RDF xml:lang="en"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">

<rdf:Description rdf:ID="MotorVehicle">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="http://www.w3.org/2000/01/rdf-schema#Resource"/>
</rdf:Description>

<rdf:Description rdf:ID="PassengerVehicle">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="Truck">
   <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
   <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

</RDF>
```

RDF Schema Namespace

An rdf:ID attribute names a **new** resource

("Resource" is the top level class)

PassengerVehicle is a **subclass** of MotorVehicle

RDF Schema Example (cont..)

```
<rdf:Description rdf:ID="Van">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#MotorVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="MiniVan">
  <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
  <rdfs:subClassOf rdf:resource="#Van"/>
  <rdfs:subClassOf rdf:resource="#PassengerVehicle"/>
</rdf:Description>

<rdf:Description rdf:ID="registeredTo">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle"/>
  <rdfs:range rdf:resource="#Person"/>
</rdf:Description>

<rdf:Description rdf:ID="price">
  <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
  <rdfs:domain rdf:resource="#MotorVehicle "/>
  <rdfs:domain rdf:resource="#Book "/>
  <rdfs:range rdf:resource="http://www.w3.org/2000/03/example/classes#Number"/>
</rdf:Description>
```

Multiple Inheritance

Domain of a property

Range of a property

Multiple Domains

http://www.w3.org/TR/rdf-mt/#RDFSRules

RDFS Entailment (1)

RDFS enables some reasoning with RDF and deduce additional information on an RDF graph

- Resource typing

(?x rdf:type ?c) → (?c rdf:type rdfs:Class)

(?x ?p ?y) → (?p rdf:type rdf:Property)

  - :LosAngeles rdf:type dbpedia:City → dbpedia:City rdf:type:Class
  - :LosAngeles ex:hasPopulation "3.8M" → ex:HasPopulation rdf:type rdf:Property

- Transitivity

(?c1 rdfs:subClassOf ?c2) ^ (?c2 rdfs:subclassOf ?c3) → (?c1 rdfs:subClassOf ?c3)

(?p1 rdfs:subPropertyOf ?p2) ^ (?p2 rdfs:subPropertyOf ?p3) →
(?p1 rdfs:subPropertyOf ?p3)

  - :Human rdfs:subClassOf :Mammal ^ :Mammal rdfs:subClassOf :Animal
    → :Human rdfs:subClassOf :Animal
  - :hasSon rdfs:subPropertyOf :hasChild ^ :hasChild rdfs:subPropertyOf :hasDescendant
    → :hasChild rdfs:subPropertyOf :hasDescendant

http://www.w3.org/TR/rdf-mt/#RDFSRules

RDFS Entailment (2)

RDFS enables some reasoning with RDF and deduce additional information on an RDF graph

- Inheritance

(?x rdf:type ?c1) ^ (?c1 rdfs:subClassOf ?c2) → (?x rdf:type ?c2)

(?x ?p1 ?y) ^ (?p1 rdfs:subPropertyOf ?p2) → (?x ?p2 ?y)

  - :John rdf:type :Human ^ :Human rdfs:subClassOf :Animal → :John rdf:type :Animal

- Domain/Range typing

(?x ?p ?y) ^ (?p rdfs:domain ?c) → (?x rdf:type ?c)

(?x ?p ?y) ^ (?p rdfs:range ?c) → (?y rdf:type ?c)

  - :John :drives :BMW1 ^ :drives rdfs:range :Car → :BMW1 rdf:type :Car

RDF & RDFS

- RDF provides a foundation for representing and processing data and metadata
- RDF has a graph-based data model
- RDF has a decentralized philosophy and allows incremental building of knowledge, and its sharing and reuse
- RDF Schema provides a mechanism for describing specific domains
- RDF Schema is a minimalist ontology language
  - instances, classes, properties, subclasses, subproperties, domain and range restrictions
- SPARQL is the query languages for RDF

Limitations of the Expressive Power of RDF Schema (1)

- No disjointness of classes
  - Sometimes we wish to say that classes are disjoint
  - Ex: male and female are disjoint
  - Ex: a person is not a city (open world assumption)
- No boolean combinations of classes
  - Sometimes we wish to build new classes by combining other classes using union, intersection, and complement
  - Ex: person is the union of classes man and woman

Limitations of the Expressive Power of RDF Schema (2)

- No cardinality restrictions
  - Ex: a person has exactly two parents,
  - Ex: a course is taught by at least one lecturer
- No special characteristics of properties
  - Transitive property (like “greaterThan”)
  - Unique property (like “isMotherOf”)
  - Inverse properties: “eats” and “isEatenBy”

Limitations of the Expressive Power of RDF Schema (3)

- No local scoping of properties
  - rdfs:range defines the range of a property (e.g. eats) for all classes
  - In RDF Schema we cannot declare range restrictions that apply to some classes only
  - E.g. we cannot say that cows eat only plants, while other animals may eat meat, too

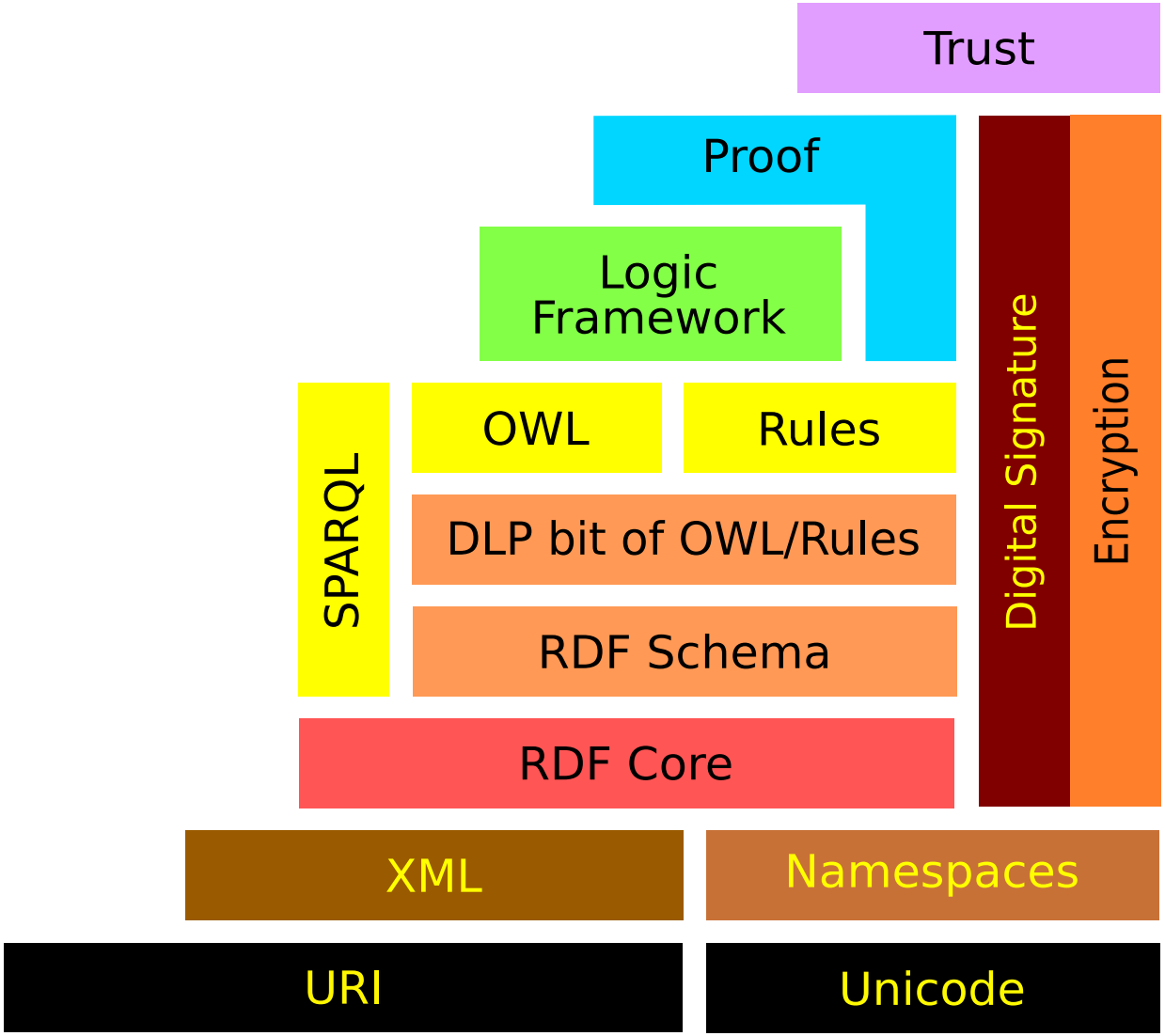
Limitations of RDFS

- RDF Schema is a simple modelling language:
  - Cannot define: Boolean combinations of classes, disjointness, cardinality restrictions, ..
  - Cannot define properties of properties: unique, inverse, transitive, …
  - No mechanism of specifying necessary and sufficient conditions for class membership.
    - Ex: If John has a vehicle which is 7ft high, has wide wheels and loading space is 3 m^3, then we should be able to reason that John has a SUV, if the necessary and sufficient conditions for a vehicle being an SUV are height > 6ft & wide wheels & loading space > 2 m^3
- Therefore we need an ontology and reasoning layer on top of RDF and RDF Schema → OWL

OWL vs RDFS

- Ideally, OWL would extend RDF Schema to be consistent with the layered architecture of the Semantic Web
- However, directly combining RDF Schema with classical logic leads to problems:
  - RDF allows resources to be instances and classes (elements and sets) at the same time, which leads to paradoxes.
  - RDF/RDFS was formalized using non-standard semantics:
    - RDF Semantics: http://www.w3.org/TR/rdf-mt/

Semantic Web Layer Cake



slide by Pedro Szekely

OWL: Web Ontology Language

- Goal: provide expressive language for knowledge representation on the web
  - Address limitations of RDFS
- Description logic substrate
  - Concepts, roles, instances
  - Concept constructors
  - Decidable subsets of first-order logic
- OWL1: W3C Recommendation 10 February 2004
  - OWL1 defined 3 different sublanguages: OWL Full, OWL DL, OWL Lite
- OWL2: W3C Recommendation 27 October 2009
  - Inherits OWL1 languages, extends OWL DL
  - Introduces 3 new profiles with "good" computational properties:
    - OWL2 QL, OWL2 EL, OWL2 RL

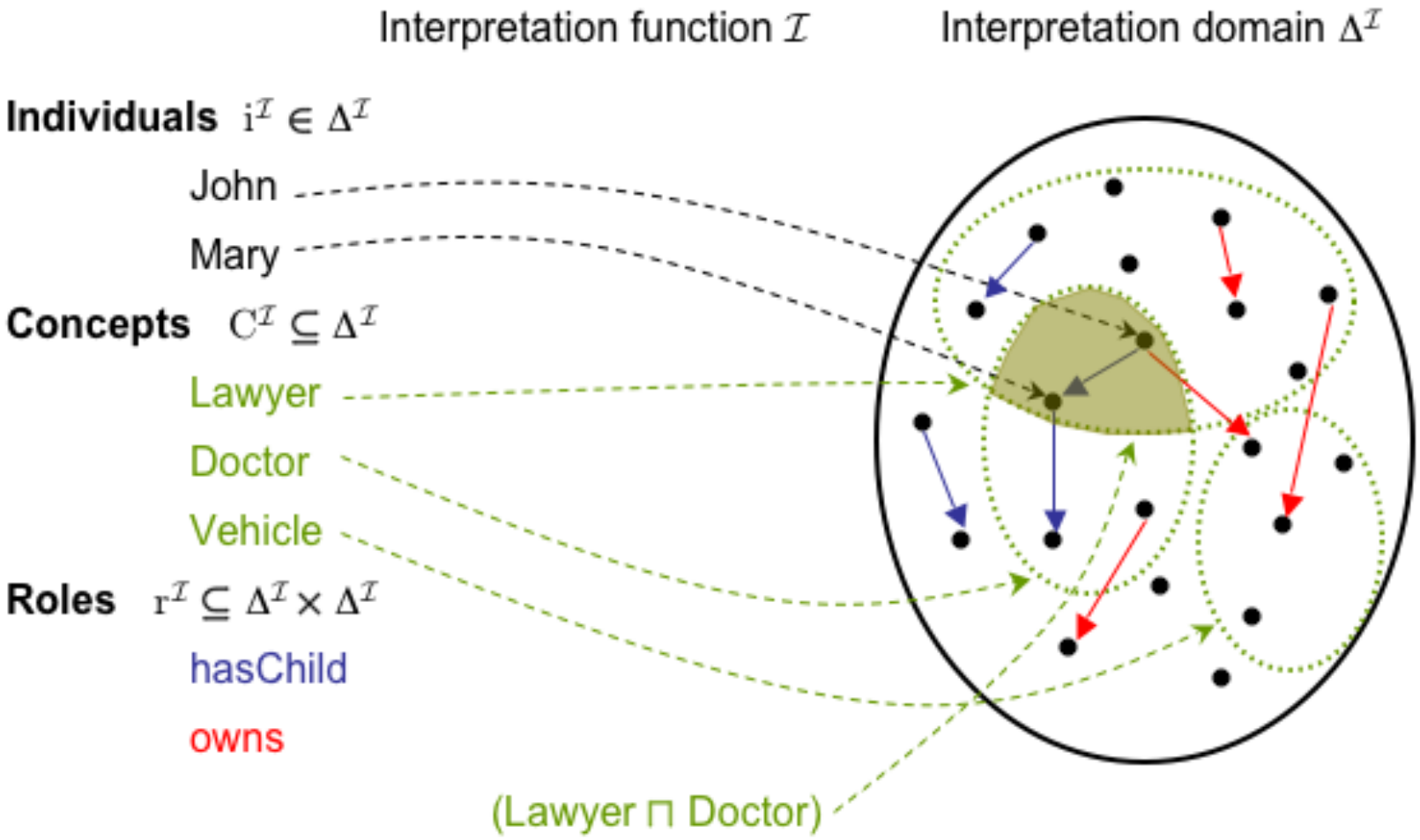
http://www.w3.org/TR/owl2-overview
http://www.w3.org/TR/owl2-primer
http://www.w3.org/TR/owl2-quick-reference/

Description Logic: Basic Ideas

- Individual = Instance = object of the domain of discourse = logical constant
  - Ex: John, Mary, Italy
- Concept = Class = set of objects = unary predicate/formula with one free variable p(x)
  - Ex: Person, Doctor, HappyParent, (Doctor $\wedge$ Lawyer)
- Role = Property = relationship between two objects = set of pairs of objects from the domain of discourse = binary predicate/formula with two free variables r(x,y)
  - Ex: hasChild, likes, worksFor

Description Logic Semantics

- Semantics given by standard first-order



Abox, Tbox, Rbox

- Abox: all the fact assertions
  - Similar to the data of a classical DB
  - But, inferences can add assertions to Abox
- Tbox (Rbox): all the terminological and relationship assertions
  - Similar to the schema of a classical DB
  - But, inferences can generate new axioms
  - Axioms are not just constraints to check, they assert new knowledge

Fact Assertions

- Concept assertion: assign a class to an individual
  - Mother(julia)
- Role assertion: asserting a relationship between two individuals
  - hasChild(julia, john)
- Inequality assertion
  - julia $\neq$ john
- Equality assertion (owl:sameAs)
  - john $\approx$ johnny

Terminological Knowledge: Concepts

- Concept Inclusion = subclassOf = set inclusion
  - $A \sqsubseteq B$: the set of objects denoted by A is a subset of the set of objects denoted by B
  - Ex: Human $\sqsubseteq$ Animal
  - $\forall x$ Human(x) $\rightarrow$ Animal(x)
- Concept Equivalence = set equality
  - $A \equiv B$: two concepts have the same instances
  - Ex: Person $\equiv$ Human
  - $\forall x$ Human(x) $\leftrightarrow$ Person(x)

Terminological Knowledge: Relationships

- Role Inclusion = set inclusion (elements are pairs of objects)
  - Ex: manages $\sqsubseteq$ worksFor$^-$
- Role composition
  - brotherOf $\circ$ parentOf $\sqsubseteq$ uncleOf
  - $\forall$ x,y brotherOf(x,y) ^ parentOf(y,z) $\rightarrow$ uncleOf(x,z)
- Role Equivalence = set equivalence
  - Ex: worksFor $\equiv$ isEmployeeOf
  - $\forall$ x, y worksFor(x,y) $\leftrightarrow$ isEmployeeOf(x,y)

Open-world vs Closed-world

- Open-World Assumption: cannot conclude some statement x to be false simply because we cannot show x to be true
  - Typical of logic and knowledge representation languages
- Closed-World Assumption:
  if we cannot prove x, we assume it is false
  - Typical of databases
- Examples:
  - Question: "Did it rain in Tokyo yesterday?"
    - Open-world Answer: "I don't know that it rained, but that's not enough reason to conclude that it didn't rain"
    - Closed-world answer: "I don't have a record that it rained, so I conclude that it didn't rain"
  - Question: "Does John work at IBM?"
    - Closed-world : there is no record for John in the employee table, so he doesn't
    - Open-world: I don't know (the database may be incomplete)

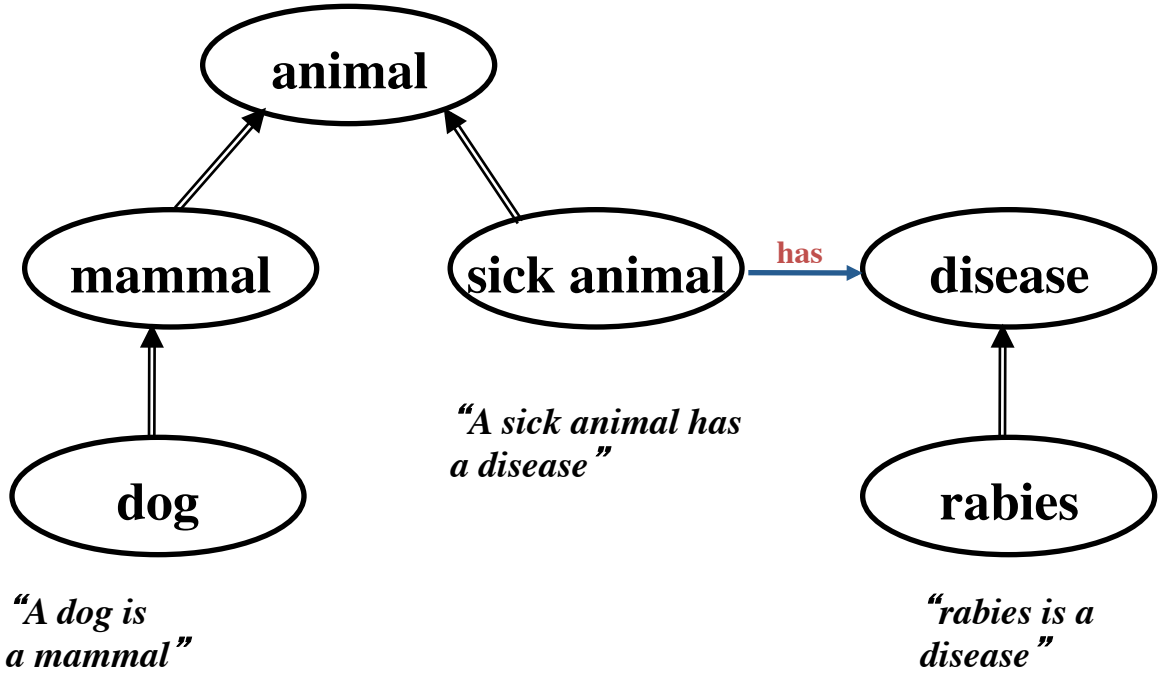
Unique-name assumption (UNA)

- UNA: When two individuals have different names, they are in fact different individuals
  - Typical of database systems
- This assumption sometimes works (ex. Product codes) and sometimes does not (ex. People)
- OWL does not make the unique-name assumption

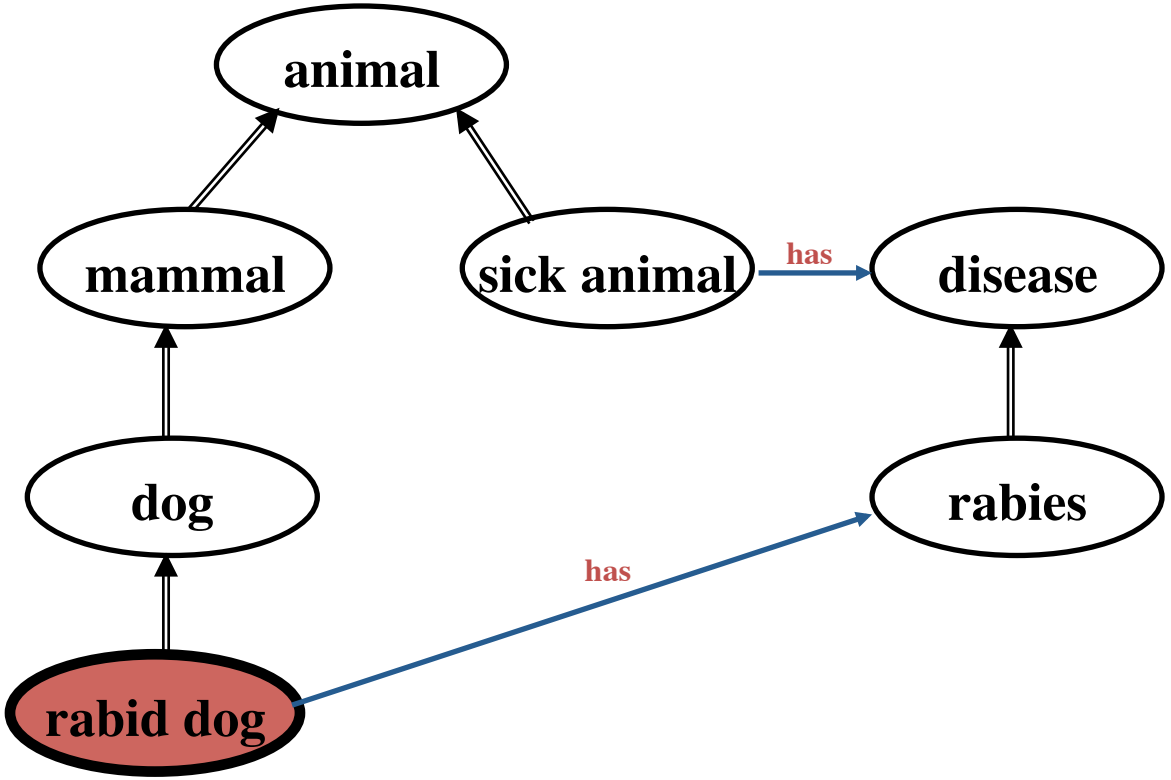
Description Logics

- Classes are defined in terms of other classes/ relations
- Powerful inference algorithms:
  - Subsumption: is class A a subclass of class B given their definitions?
  - Recognition: is x an instance of class A?
  - Classification: automatic reorganization of class hierarchy based on definitions of classes

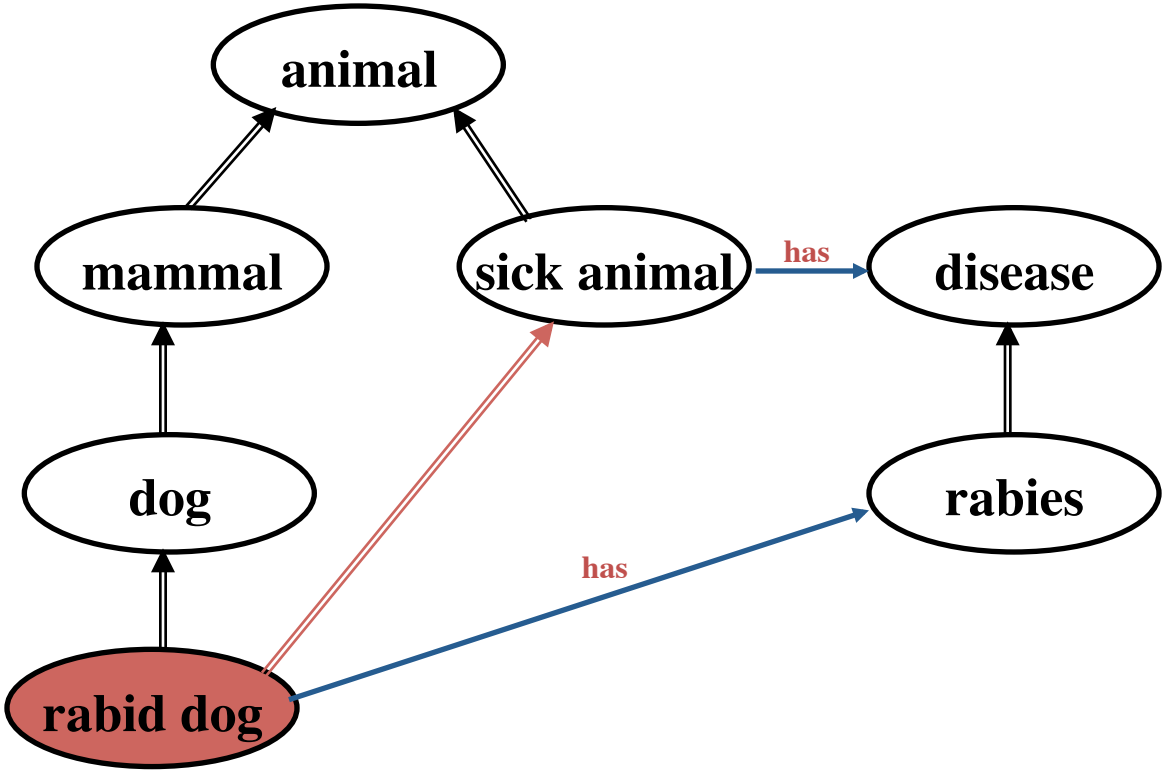
Classification: Defining an Ontology



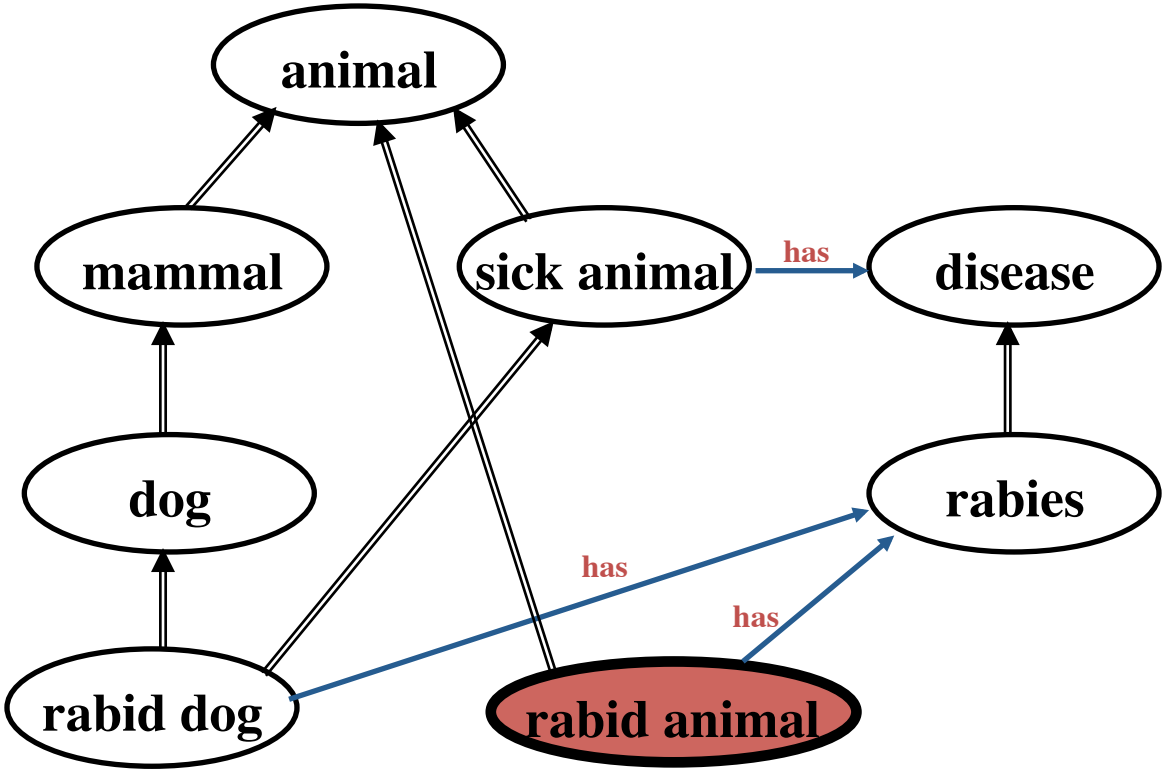
Classification: Defining a "rabid dog"



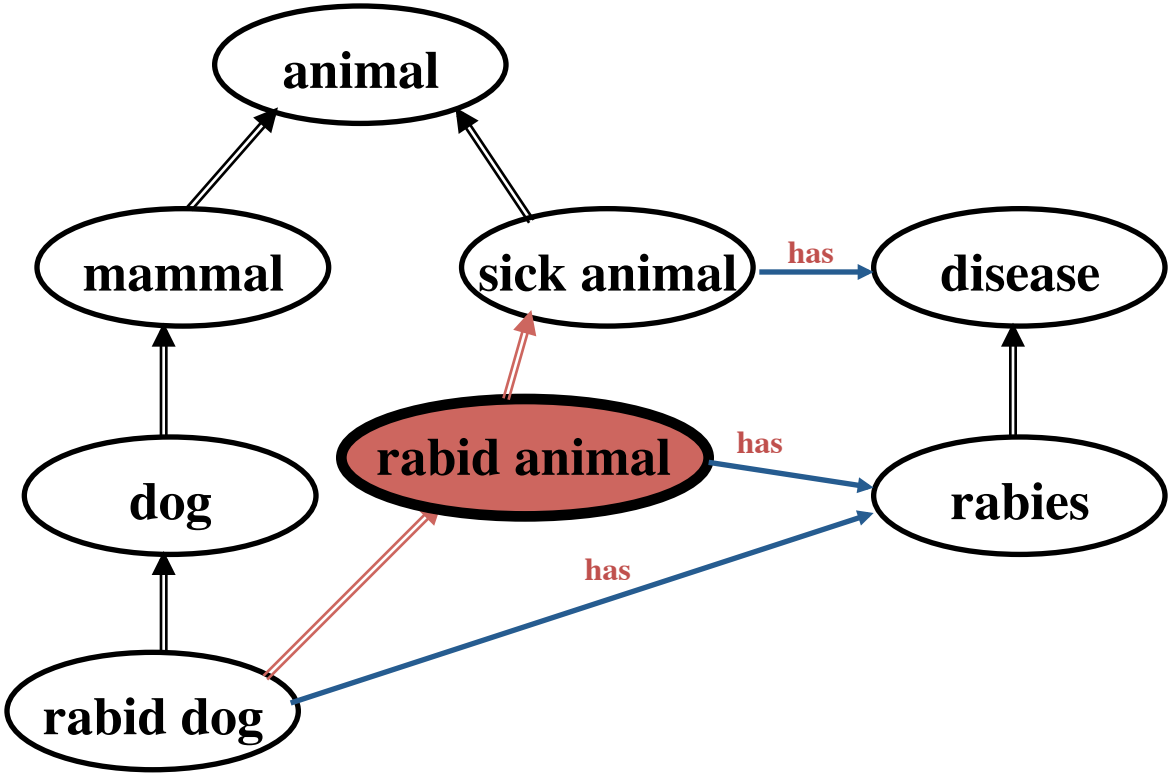
Classification:
Classifier Infers "sick animal"



Classification: Defining "rabid animal"



Classification:
Concept Placed in Hierarchy



OWL2 Description Logic

- OWL DL corresponds to a well-studied **decidable** description logic SROIQ(D)
  - 2NExpTime-complete
  - implemented reasoners: Fact++, Hermit, Pellet, RacerPro

- Lose full compatibility with RDF
  - Not every RDF document is a legal OWL DL document.
  - Every legal OWL DL document is a legal RDF document.

OWL2 DL Summary

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

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

based on slide from Pascal Hitzler

OWL Classes

- Classes are defined using **owl:Class**
  - owl:Class is a subclass of rdfs:Class
- **owl:Thing** is the most general class, which contains everything (T)
  - ~ True
- **owl:Nothing** is the empty class (⊥)
  - ~ False

OWL Properties

- **Object properties**: relate objects to other objects
  - Ex: isTaughtBy, supervises, …

```
<owl:ObjectProperty rdf:ID="teaches">
   <rdfs:domain rdf:resource="#teacher"/>
   <rdfs:range rdf:resource= "#course"/>
   <owl:inverseOf rdf:resource="#isTaughtBy"/>
</owl:ObjectProperty>
```

$\exists$ teaches.T $\sqsubseteq$ Teacher

T $\sqsubseteq$ $\forall$ teaches.Course

- **Datatype properties**: relate objects to data values (XML Schema datatypes)
  - Ex: phone, title, age, …

```
<owl:DatatypeProperty rdf:ID="age">
   <rdfs:range rdf:resource="http://www.w3.org/2001/XLMSchema
      #nonNegativeInteger"/>
</owl:DatatypeProperty>
```

Ontology Axioms

OWL Syntax	DL Syntax	Example
rdfs: subClassOf	$C_1 \sqsubseteq C_2$	Human $\sqsubseteq$ Animal $\sqcap$ Biped
owl: equivalentClass	$C_1 \equiv C_2$	Man $\equiv$ Human $\sqcap$ Male
rdfs: subPropertyOf	$P_1 \sqsubseteq P_2$	hasDaughter $\sqsubseteq$ hasChild
owl: equivalentProperty	$P_1 \equiv P_2$	cost $\equiv$ price
owl: transitiveProperty	$P^+ \sqsubseteq P$	ancestor$^+$ $\sqsubseteq$ ancestor

OWL Syntax	DL Syntax	Example
rdf: type	$a : C$	John : Happy-Father
property	$\langle a, b \rangle : R$	$\langle$John, Mary$\rangle$: has-child

owl:ObjectProperty
owl:DatatypeProperty

Class Definitions

- In description logics and OWL, classes can be:
  - Primitive: just a name for a set of objects
  - Defined: logical formula using class constructors
- Concept/Role constructors:
  - Capture basic representational features
    - 30 year history of knowledge representation
  - Chosen so that inference (e.g., satisfiability/subsumption) is decidable and, if possible, of low complexity
- Two main types of class constructors:
  - Restrictions on properties
  - Logical Boolean operators: and, or, not (complement)
- Many description logics, depending on choice of concept/role constructors

Class/Concept Constructors

Constructor	DL Syntax	Example	FOL Syntax
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male	$C_1(x) \wedge \ldots \wedge C_n(x)$
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer	$C_1(x) \vee \ldots \vee C_n(x)$
complementOf	$\neg C$	$\neg$Male	$\neg C(x)$
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}	$x = x_1 \vee \ldots \vee x = x_n$
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor	$\forall y.P(x,y) \rightarrow C(y)$
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer	$\exists y.P(x,y) \wedge C(y)$
maxCardinality	$\leqslant nP$	$\leqslant$1hasChild	$\exists^{\leqslant n} y.P(x,y)$
minCardinality	$\geqslant nP$	$\geqslant$2hasChild	$\exists^{\geqslant n} y.P(x,y)$

- C is a concept (class); P is a role (property);
- x, y are variables ranging over individuals

OWL2 is essentially the Description Logic *SROIQ*

$$\mathsf{R} ::= \mathsf{U} \mid \mathsf{N}_{\mathsf{R}} \mid \mathsf{N}_{\mathsf{R}^-}$$

$$\mathbf{C} ::= \mathsf{N}_C \mid (\mathbf{C} \sqcap \mathbf{C}) \mid (\mathbf{C} \sqcup \mathbf{C}) \mid \neg \mathbf{C} \mid \top \mid \bot \mid$$

$$\exists \mathbf{R}.\mathbf{C} \mid \forall \mathbf{R}.\mathbf{C} \mid \geqslant n\, \mathbf{R}.\mathbf{C} \mid \leqslant n\, \mathbf{R}.\mathbf{C} \mid \exists \mathbf{R}.\mathit{Self} \mid \{\mathsf{N}_I\}$$

ABox:	$\mathbf{C}(\mathsf{N}_I)$	$\mathbf{R}(\mathsf{N}_I, \mathsf{N}_I)$	$\mathsf{N}_I \approx \mathsf{N}_I$	$\mathsf{N}_I \not\approx \mathsf{N}_I$
TBox:	$\mathbf{C} \sqsubseteq \mathbf{C}$	$\mathbf{C} \equiv \mathbf{C}$		
RBox:	$\mathbf{R} \sqsubseteq \mathbf{R}$	$\mathbf{R} \equiv \mathbf{R}$	$\mathbf{R} \circ \mathbf{R} \sqsubseteq \mathbf{R}$	$\mathit{Disjoint}(\mathbf{R}, \mathbf{R})$

OWL Property Restrictions

- In OWL we can define a class by restricting the way it participates in a property
- A (restriction) class is achieved through an **owl:Restriction** element, which contains
  - **owl:onProperty** element
  - one or more restriction declarations
- Value Restrictions: restrictions on the kinds of values a property may take
  - **owl:someValuesFrom**
  - **owl:allValuesFrom**
  - **owl:hasValue**
- Cardinality Restrictions: restrictions on the number of individuals participating in the relation
  - **owl:minCardinality** (at least)
  - **owl:maxCardinality** (at most)
  - **owl:cardinality** (exactly)

owl:someValuesFrom

Qualified existential restriction: $\exists$ teaches.Course
The set of individuals that teach some (at least 1) course

Ex: Teacher $\sqsubseteq$ $\exists$ teaches.Course

```
<owl:Class rdf:about=”#Teacher">
  <rdfs:subClassOf>
    <owl:Restriction>
        <owl:onProperty rdf:resource="#teaches"/>
        <owl:someValuesFrom rdf:resource=“#Course"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

owl:allValuesFrom

Qualified universal restriction: $\forall$ isTaughtBy.Professor

The set of things that are taught only by professors

Ex: UniversityCourse $\sqsubseteq$ $\forall$ isTaughtBy.Professor

```
<owl:Class rdf:about="#UniversityCourse">
   <rdfs:subClassOf>
     <owl:Restriction>
         <owl:onProperty rdf:resource="#isTaughtBy"/>
         <owl:allValuesFrom rdf:resource="#Professor"/>
     </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Nominals: owl:hasValue

Nominal = concept that has only one instance

Ex: {Germany} is the concept containing only the individual Germany

Ex: German $\sqsubseteq \exists$ bornIn.{Germany}

A German is an individual born in Germany

```
<owl:Class rdf:about=”#German">
   <rdfs:subClassOf>
      <owl:Restriction>
            <owl:onProperty rdf:resource="#bornIn"/>
            <owl:hasValue rdf:resource= "#Germany"/>
            </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

Nominals: Enumerated classes with owl:oneOf

DaysOfTheWeek ≡ {Monday} ⊔ … ⊔ {Sunday}

```
<owl:Class rdf:ID="DaysOfTheWeek">
    <owl:oneOf rdf:parseType="Collection">
        <owl:Thing rdf:about="#Monday"/>
        <owl:Thing rdf:about="#Tuesday"/>
        <owl:Thing rdf:about="#Wednesday"/>
        <owl:Thing rdf:about="#Thursday"/>
        <owl:Thing rdf:about="#Friday"/>
        <owl:Thing rdf:about="#Saturday"/>
        <owl:Thing rdf:about="#Sunday"/>
    </owl:oneOf>
</owl:Class>
```

Cardinality Restrictions (1)

Unqualified Cardinality constructor: individuals that have some number of values for a role

- Ex: ≥ 1 hasChild

  The set of individuals that have at least one child
- Ex: Parent $\sqsubseteq \geq 1$ hasChild

  Parents are individuals with at least one child

```
<owl:Class rdf:about=”#Parent">
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hasChild"/>
           <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">
              1
           </owl:minCardinality>
     </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Note: ≥ 1 r implies $\exists$ r

Cardinality Restrictions (2)

Qualified Cardinality constructor: individuals that have some number of values *of a given type* for a role

- Ex: $\geqslant$ 1 hasChild.Smart

  The set of individuals that have at least one child that is smart

- Ex: HappyParent $\sqsubseteq$ $\geqslant$ 1 hasChild.Smart

  Happy parents are individuals with at least one child that is smart

```
<owl:Class rdf:about=”#Parent">
   <rdfs:subClassOf>
     <owl:Restriction>
       <owl:onProperty rdf:resource="#hasChild"/>
            <owl:minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">
              1
            </owl:minQualifiedCardinality>
       <owl:onClass rdf:resource=“#Smart”/>
     </owl:Restriction>
   </rdfs:subClassOf>
</owl:Class>
```

Local reflexivity: Self

Individuals related to themselves through a role

```
<owl:Class rdf:about="NarcisticPerson”>
  <owl:equivalentClass>
    <owl:Restriction>
       <owl:onProperty rdf:resource="loves"/>
       <owl:hasSelf rdf:datatype=”&xsd;#boolean"> true </owl:hasSelf>
    </owl:Restriction>
  </owl:equivalentClass>
</owl:Class>
```

Boolean Operators: Conjunction (Intersection)

FacultyInCS ≡ Faculty ⊓ ∃ belongsTo.{CSDepartment}

```
<owl:Class rdf:ID="FacultyInCS">
  <owl:equivalentClass>
    <owl:intersectionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#Faculty"/>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#belongsTo"/>
        <owl:hasValue rdf:resource="#CSDepartment"/>
      </owl:Restriction>
    </owl:intersectionOf>
  </owl:equivalentClass>
</owl:Class>
```

Boolean Operators: Disjunction (Union)

UniversityPeople ≡ Faculty ⊔ Staff ⊔ Student

```
<owl:Class rdf:ID="UniversityPeople">
   <owl:equivalentClass>
    <owl:unionOf rdf:parseType="Collection">
        <owl:Class rdf:about=”#Faculty"/>
        <owl:Class rdf:about=”#Staff"/>
        <owl:Class rdf:about=”#Student"/>
    </owl:unionOf>
  <owl:equivalentClass>
</owl:Class>
```

Boolean Operators: Negation (Complement)

```
<owl:Class rdf:about=”#Vegetarian">
      <rdfs:subClassOf>
          <owl:Restriction>
              <owl:complementOf rdf:resource=”#Carnivorous"/>
          </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

$\text{Vegetarian} \sqsubseteq \neg\ \text{Carnivorous}$

```
<owl:Class rdf:about=”#Course">
      <rdfs:subClassOf>
          <owl:Restriction>
              <owl:complementOf rdf:resource=”#Professor"/>
          </owl:Restriction>
      </rdfs:subClassOf>
</owl:Class>
```

$\text{Course} \sqsubseteq \neg\ \text{Professor}$

Property Characteristics

- owl:TransitiveProperty
  - Ex: “has better grade than”, “is ancestor of”
- owl:SymmetricProperty
  - Ex: “has same grade as”, “is sibling of”
- owl:FunctionalProperty defines a property that has at most one value for each object
  - Ex: “age”, “height”, “directSupervisor”
- owl:InverseFunctionalProperty defines a property for which two different objects cannot have the same value

Class Equivalence

- Faculty ≡ AcademicStaff

```
<owl:Class rdf:ID=“Faculty">
   <owl:equivalentClass  rdf:resource=”#AcademicStaff"/>
</owl:Class>
```

Class Disjointness

- Male $\sqsubseteq \neg$ Female

```
<owl:Class rdf:about=”#Male”>
    <owl:disjointWith rdf:resource=”#Female"/>
</owl:Class>
```

- AssistantProfessor $\sqsubseteq \neg$ AssociateProfessor

  AssistantProfessor $\sqsubseteq \neg$ Professor

  AssociateProfessor $\sqsubseteq \neg$ Professor

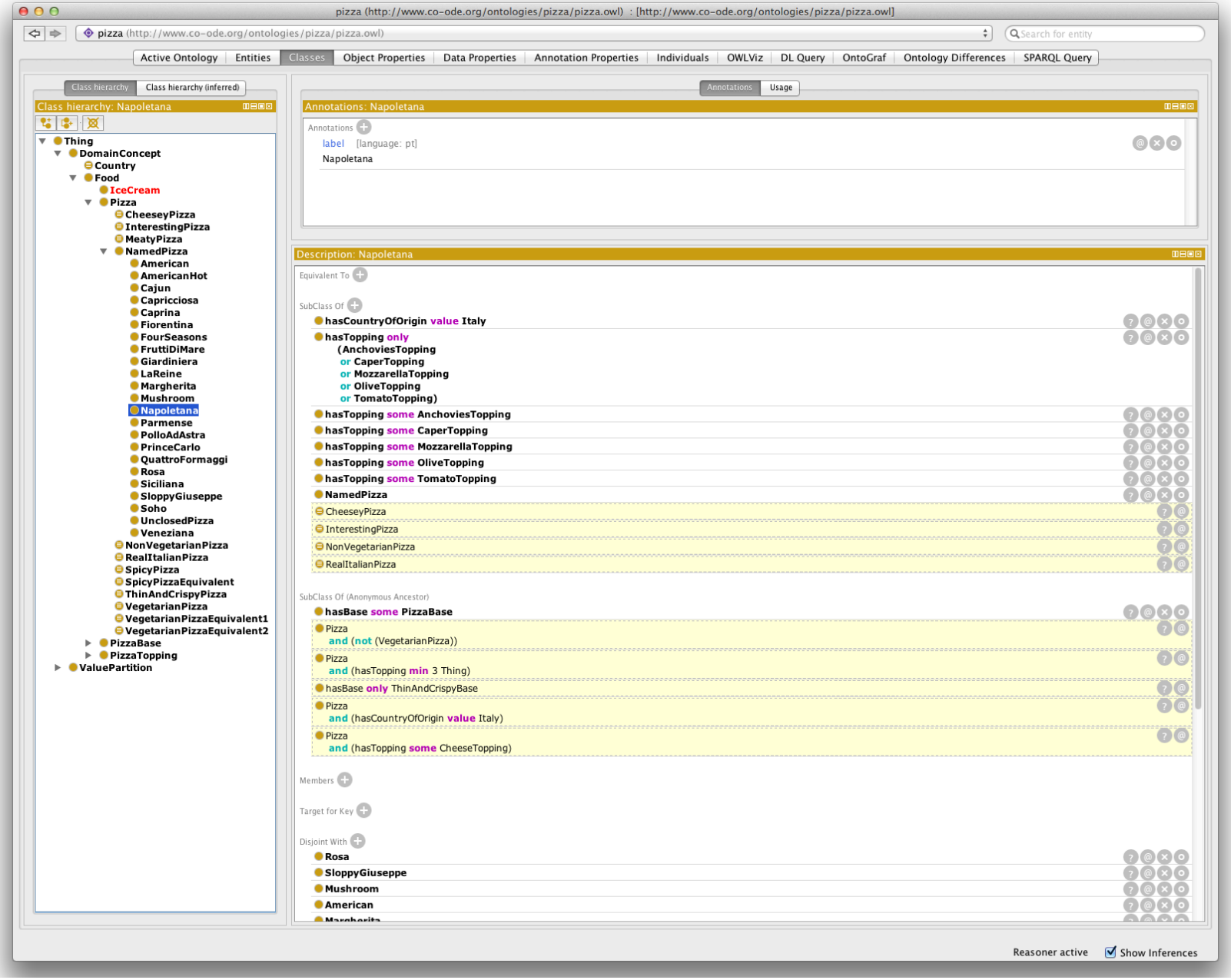
```
<owl:AllDisjointClasses>
    <owl:members rdf:parseType="Collection">
        <owl:Class rdf:about=”#AssistantProfessor ">
        <owl:Class rdf:about=”#AssociateProfessor">
        <owl:Class rdf:about=”#Professor”>
    </owl:members>
</owl:AllDisjointClasses>
```

OWL RDF/XML Exchange Syntax

$\text{Person} \sqcap \forall \text{hasChild}.(\text{Doctor} \sqcup \exists \text{hasChild}.\text{Doctor}))$

```
<owl:Class>
 <owl:intersectionOf rdf:parseType=“ collection">
  <owl:Class rdf:about="#Person"/>
  <owl:Restriction>
   <owl:onProperty rdf:resource="#hasChild"/>
   <owl:allValuesFrom>
    <owl:unionOf rdf:parseType=" collection">
     <owl:Class rdf:about="#Doctor"/>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#hasChild"/>
      <owl:someValuesFrom rdf:resource="#Doctor"/>
     </owl:Restriction>
    </owl:unionOf>
   </owl:allValuesFrom>
  </owl:Restriction>
 </owl:intersectionOf>
</owl:Class>
```

Protégé


pizza (http://www.co-ode.org/ontologies/pizza/pizza.owl) : [http://www.co-ode.org/ontologies/pizza/pizza.owl]
pizza (http://www.co-ode.org/ontologies/pizza/pizza.owl)
Search for entity
Active Ontology
Entities
Classes
Object Properties
Data Properties
Annotation Properties
Individuals
OWLViz
DL Query
OntoGraf
Ontology Differences
SPARQL Query
Class hierarchy
Class hierarchy (inferred)
Class hierarchy: Napoletana
Thing
DomainConcept
Country
Food
IceCream
Pizza
CheeseyPizza
InterestingPizza
MeatyPizza
NamedPizza
American
AmericanHot
Cajun
Capricciosa
Caprina
Fiorentina
FourSeasons
FruttiDiMare
Giardiniera
LaReine
Margherita
Mushroom
Napoletana
Parmense
PolloAdAstra
PrinceCarlo
QuattroFormaggi
Rosa
Siciliana
SloppyGiuseppe
Soho
UnclosedPizza
Veneziana
NonVegetarianPizza
RealItalianPizza
SpicyPizza
SpicyPizzaEquivalent
ThinAndCrispyPizza
VegetarianPizza
VegetarianPizzaEquivalent1
VegetarianPizzaEquivalent2
PizzaBase
PizzaTopping
ValuePartition
Annotations
Usage
Annotations: Napoletana
Annotations
label [language: pt]
Napoletana
Description: Napoletana
Equivalent To
SubClass Of
hasCountryOfOrigin value Italy
hasTopping only
(AnchoviesTopping
or CaperTopping
or MozzarellaTopping
or OliveTopping
or TomatoTopping)
hasTopping some AnchoviesTopping
hasTopping some CaperTopping
hasTopping some MozzarellaTopping
hasTopping some OliveTopping
hasTopping some TomatoTopping
NamedPizza
CheeseyPizza
InterestingPizza
NonVegetarianPizza
RealItalianPizza
SubClass Of (Anonymous Ancestor)
hasBase some PizzaBase
Pizza
and (not (VegetarianPizza))
Pizza
and (hasTopping min 3 Thing)
hasBase only ThinAndCrispyBase
Pizza
and (hasCountryOfOrigin value Italy)
Pizza
and (hasTopping some CheeseTopping)
Members
Target for Key
Disjoint With
Rosa
SloppyGiuseppe
Mushroom
American
Reasoner active
Show Inferences

Inference Problems

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

based on slide from Pascal Hitzler

Inference Problems: Reduction to Unsatisfiability

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

based on slide from Pascal Hitzler

Tableaux Inference Algorithms

Reduction to (un)satisfiability.

Idea:

- Given knowledge base K
- Construct a tree (called *Tableau,* actually a forest), which represents a model of K.
- If attempt fails, K is unsatisfiable.

based on slide from Pascal Hitzler

Tableaux rules

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

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

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

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

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

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

based on slide from Pascal Hitzler

Tableaux example 1

KB:

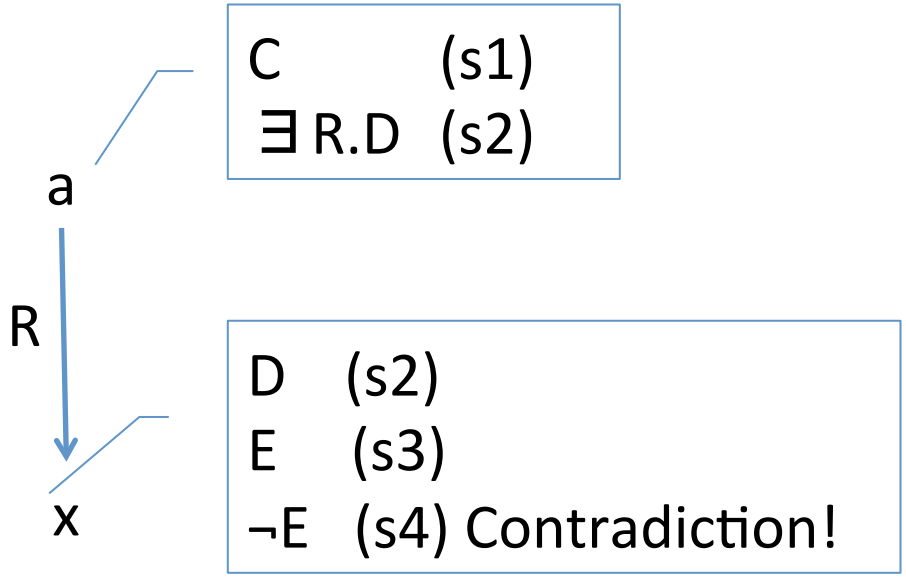
s1: C(a)

s2: $C \sqsubseteq \exists R.D$

s3: $D \sqsubseteq E$

Query: Does KB entail

$(\exists R.E)(a)$?



Approach:

1. Add negation of query:

   s4: $(\forall R.\neg E)(a)$

   to KB

2. Show unsatisfiability

based on slide from Pascal Hitzler

Tableaux example 2 Reasoning by Cases

KB:

s1: C(a)

s2: $C \sqsubseteq \exists R.D$

s3: $D \sqsubseteq E \sqcup F$

s4: $F \sqsubseteq E$

Query: Does KB entail

$(\exists R.E)(a)$?

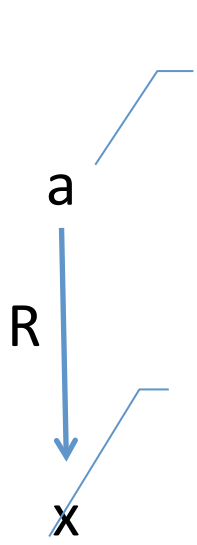
Approach:

1. Add negation of query:

   s5:$(\forall R.\neg E)(a)$

   to KB

2. Show unsatisfiability



C (s1)
$\exists R.D$ (s2)

D (s2)
$\neg E$ (s5)
Choice: $D \sqsubseteq E \sqcup F$ (s3)
1. E Contradiction!
2. F
   E (s4) Contradiction!

based on slide from Pascal Hitzler

Resources for OWL and DL

- Description Logic Handbook, Cambridge University Press
  - http://books.cambridge.org/0521781760.htm
- Description Logic: http://dl.kr.org/
  - complexity: http://www.cs.man.ac.uk/~ezolin/dl
- Web Ontology Language (OWL):
  - http://www.w3.org/TR/owl2-overview
- Reasoners:
  - Pellet (open source): http://pellet.owldl.com/
  - FaCT++ (open source): http://owl.man.ac.uk/factplusplus/
  - Racer (comercial): http://www.racer-systems.com/
  - (Loom and Powerloom: http://www.isi.edu/isd/LOOM/)
- Ontology Editors:
  - Protégé: http://protege.stanford.edu/
- Ian Horrocks has great slides on description logics and OWL:
  - http://web.comlab.ox.ac.uk/oucl/work/ian.horrocks/

Additional Slides

REASONING

$\sqcap$ rule:

$\sqcup$ rule:

$\exists$ rule:

$\forall$ rule:

A-box : our triples

Example

$A = \{ a:C_1 , b:C_2 \}$

an A-box with 2 triples

REASONING

$\sqcap$ rule: $a : (C_1 \sqcap C_2) \in A$

what else goes in the A-box?

$\sqcup$ rule:

$\exists$ rule:

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a : (C_1 \sqcap C_2) \in A$ and

2. $\{a : C_1, a : C_2\} \not\subseteq A$

$\sqcup$ rule: then ...

$\exists$ rule:

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a : (C_1 \sqcap C_2) \in A$ and

2. $\{a : C_1, a : C_2\} \notin A$

$\sqcup$ rule: then set $A_1 = A \cup \{a : C_1, a : C_2\}$

$\exists$ rule: $A = \{ a : \text{Engineer} \sqcap \text{Artist} \}$

$A_1 =$

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a : (C_1 \sqcap C_2) \in A$ and

2. $\{a : C_1, a : C_2\} \notin A$

$\sqcup$ rule: then set $A_1 = A \cup \{a : C_1, a : C_2\}$

$\exists$ rule:

Example

$A = \{ a : \text{Engineer} \sqcap \text{Artist} \}$

$\forall$ rule:

$$A_1 = \left\{ \begin{array}{l} a : \text{Engineer}, a : \text{Artist}, \\ a : \text{Engineer} \sqcap \text{Artist} \end{array} \right\}$$

REASONING

$\sqcap$ rule: if 1. $a : (C_1 \sqcap C_2) \in A$ and
2. $\{a : C_1, a : C_2\} \not\subseteq A$
then set $A_1 = A \cup \{a : C_1, a : C_2\}$

$\sqcup$ rule: if 1. $a : (C_1 \sqcup C_2) \in A$ and
2. $\{a : C_1, a : C_2\} \cap A = \phi$
then set

$\exists$ rule:

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a:(C_1 \sqcap C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \not\subseteq A$
then set $A_1 = A \cup \{a:C_1, a:C_2\}$

$\sqcup$ rule: if 1. $a:(C_1 \sqcup C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \cap A = \phi$
then set
$A_1 = A \cup \{a:C_1\}$ and
$A_2 = A \cup \{a:C_2\}$
we branch

$\exists$ rule:

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a:(C_1 \sqcap C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \not\subseteq A$
then set $A_1 = A \cup \{a:C_1, a:C_2\}$

$\sqcup$ rule: if 1. $a:(C_1 \sqcup C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \cap A = \phi$
then set
$A_1 = A \cup \{a:C_1\}$ and
$A_2 = A \cup \{a:C_2\}$
we branch

$\exists$ rule:

$\forall$ rule:

Example

$A = \{a: \text{Engineer} \sqcup \text{Artist}\}$ then

$A_1 = \{a: \text{Engineer} \sqcup \text{Artist}, a: \text{Engineer}\}$

$A_2 = \{a: \text{Engineer} \sqcup \text{Artist}, a: \text{Artist}\}$

REASONING

$\sqcap$ rule: if 1. $a:(C_1 \sqcap C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \not\subseteq A$
then set $A_1 = A \cup \{a:C_1, a:C_2\}$

$\sqcup$ rule: if 1. $a:(C_1 \sqcup C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \cap A = \phi$
then set
$A_1 = A \cup \{a:C_1\}$ and
$A_2 = A \cup \{a:C_2\}$
we branch

$\exists$ rule: if 1. $a:(\exists P.C) \in A$ and
2. there is no b such that
$\{\langle a,b\rangle:P, b:C\} \subseteq A$
then set

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a:(C_1 \sqcap C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \not\subseteq A$

then set $A_1 = A \cup \{a:C_1, a:C_2\}$

$\sqcup$ rule: if 1. $a:(C_1 \sqcup C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \cap A = \phi$

then set
$A_1 = A \cup \{a:C_1\}$ and
$A_2 = A \cup \{a:C_2\}$

$\exists$ rule: if 1. $a:(\exists P.C) \in A$ and

2. there is no b such that

$\{\langle a,b\rangle : P, b:C\} \subseteq A$

then set

$A_1 = A \cup \{\langle a,d\rangle : P, d:C\}$

where d is new in A

$\forall$ rule:

REASONING

$\sqcap$ rule: if 1. $a:(C_1 \sqcap C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \not\subseteq A$

then set $A_1 = A \cup \{a:C_1, a:C_2\}$

$\sqcup$ rule: if 1. $a:(C_1 \sqcup C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \cap A = \phi$

then set
$A_1 = A \cup \{a:C_1\}$ and
$A_2 = A \cup \{a:C_2\}$

$\exists$ rule: if 1. $a:(\exists P.C) \in A$ and
2. there is no b such that
$\{\langle a,b\rangle : P, b:C\} \subseteq A$

then set
$A_1 = A \cup \{\langle a,d\rangle : P, d:C\}$
where d is new in A

Example

$A = \{ a:\exists \text{parent}.\text{Doctor}, b:\text{Doctor}\}$

$\forall$ rule:

$A_1 = \{ a:\exists \text{parent}.\text{Doctor}, b:\text{Doctor}, \langle a,d\rangle:\text{parent}, d:\text{Doctor}\}$

REASONING

$\sqcap$ rule: if 1. $a:(C_1 \sqcap C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \not\subseteq A$

then set $A_1 = A \cup \{a:C_1, a:C_2\}$

$\sqcup$ rule: if 1. $a:(C_1 \sqcup C_2) \in A$ and
2. $\{a:C_1, a:C_2\} \cap A = \phi$

then set
$A_1 = A \cup \{a:C_1\}$ and
$A_2 = A \cup \{a:C_2\}$

$\exists$ rule: if 1. $a:(\exists P.C) \in A$ and
2. there is no b such that
$\{\langle a,b\rangle:P, b:C\} \subseteq A$

then set
$A_1 = A \cup \{\langle a,d\rangle:P, d:C\}$
where d is new in A

$\forall$ rule: if 1. $\{a:\forall P.C, \langle a,b\rangle:P\} \subseteq A$ and
2. $b:C \notin A$

then $A_1 = A \cup \{b:C\}$

Example $A = \{a:\forall hasChild.Doctor, \langle a,b\rangle:hasChild\}$

$A_1 = \{a:\forall hasChild.Doctor, \langle a,b\rangle:hasChild, b:Doctor\}$

REASONING

$\sqcap$ rule: if 1. $a : (C_1 \sqcap C_2) \in A$ and
2. $\{a : C_1, a : C_2\} \not\subseteq A$

then set $A_1 = A \cup \{a : C_1, a : C_2\}$

$\sqcup$ rule: if 1. $a : (C_1 \sqcup C_2) \in A$ and
2. $\{a : C_1, a : C_2\} \cap A = \phi$

then set
$A_1 = A \cup \{a : C_1\}$ and
$A_2 = A \cup \{a : C_2\}$

$\exists$ rule: if 1. $a : (\exists P.C) \in A$ and
2. there is no b such that
$\{\langle a,b\rangle : P, b : C\} \subseteq A$

then set
$A_1 = A \cup \{\langle a,d\rangle : P, d : C\}$
where d is new in A

$\forall$ rule: if 1. $\{a : \forall P.C, \langle a,b\rangle : P\} \subseteq A$ and
2. $b : C \notin A$

then $A_1 = A \cup \{b : C\}$

EXAMPLE

$\mathrm{Doctor} \sqsubseteq \mathrm{Person}$

$\mathrm{Parent} \equiv \mathrm{Person} \sqcap \exists \mathrm{hasChild}.\mathrm{Person}$

$\mathrm{HappyParent} \equiv \mathrm{Parent} \sqcap \forall \mathrm{hasChild}.(\mathrm{Doctor} \sqcup \exists \mathrm{hasChild}.\mathrm{Doctor})$

translate to English

EXAMPLE

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$\mathcal{A}$: John : HappyParent,
$\langle John, Mary \rangle$: hasChild

is Mary : Doctor ?

is Mary : Doctor ?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$\mathcal{A}$ = $John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$ add the negation, and apply the rules

is Mary : Doctor ?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$\mathcal{A}$ = $John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$ add the negation, and apply the rules

is Mary : Doctor ?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$A =$ $John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

$Mary : Doctor$

$Mary : \exists hasChild.Doctor$

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$ inconsistent

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

~~$Mary : Doctor$~~

$Mary : \exists hasChild.Doctor$

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

$Mary : \exists hasChild.Doctor$

$\langle John, a \rangle : hasChild$

$a : Doctor \sqcup \exists hasChild.Doctor$

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

$Mary : \exists hasChild.Doctor$

$\langle John, a \rangle : hasChild$

$a : Doctor \sqcup \exists hasChild.Doctor$

$\langle Mary, b \rangle : hasChild$

$b : Doctor$

More inferences

$a : Person$

$b : Person$

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,
$\langle John, Mary \rangle : hasChild$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

$Mary : \exists hasChild.Doctor$

$\langle John, a \rangle : hasChild$

$a : Doctor \sqcup \exists hasChild.Doctor$

$\langle Mary, b \rangle : hasChild$

$b : Doctor$

$a : Person$

$b : Person$

We are stuck, and we didn't reach a contradiction!

we cannot infer Mary:Doctor

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

Suppose we add one more axiom

$Mary : \forall hasChild.\bot$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

$Mary : \exists hasChild.Doctor$

$\langle John, a \rangle : hasChild$

$a : Doctor \sqcup \exists hasChild.Doctor$

$\langle Mary, b \rangle : hasChild$

$b : Doctor$

$a : Person$

$b : Person$

is Mary:Doctor?

$Doctor \sqsubseteq Person$

$Parent \equiv Person \sqcap \exists hasChild.Person$

$HappyParent \equiv Parent \sqcap \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : HappyParent$,

$\langle John, Mary \rangle : hasChild$

Suppose we add one more axiom

$Mary : \forall hasChild.\bot$

$Mary : \neg Doctor$

$John : Parent$

$John : \forall hasChild.(Doctor \sqcup \exists hasChild.Doctor)$

$John : Person$

$John : \exists hasChild.Person$

$Mary : \exists hasChild.Doctor$

$\langle John, a \rangle : hasChild$

$a : Doctor \sqcup \exists hasChild.Doctor$

$\langle Mary, b \rangle : hasChild$

$b : Doctor$

$b : \bot$ YEAH! Contradiction!

$a : Person$

$b : Person$

Tableaux Inference Algorithms

Reduction to (un)satisfiability.

Idea:

- Given knowledge base K
- Construct a tree (called *Tableau,* actually a forest), which represents a model of K.
- If attempt fails, K is unsatisfiable.

based on slide from Pascal Hitzler

Negation Normal Form

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

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

based on slide from Pascal Hitzler

NNF Example

$$D \sqsubseteq (A \sqcap B) \sqcup \neg (\neg A \sqcup C)$$

$$\neg D \sqcup (A \sqcap B) \sqcup \neg (\neg A \sqcup C)$$

$$\neg D \sqcup (A \sqcap B) \sqcup (A \sqcap \neg C)$$

based on slide from Pascal Hitzler

Tableaux rules

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

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

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

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

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

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

based on slide from Pascal Hitzler

Tableaux example 1

KB:

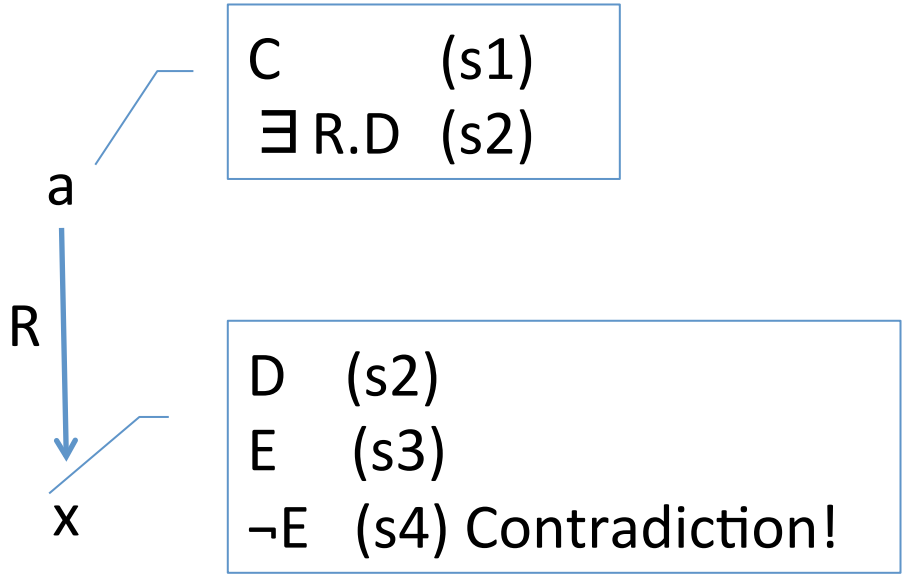
s1: C(a)

s2: $C \sqsubseteq \exists R.D$

s3: $D \sqsubseteq E$

Query: Does KB entail

$(\exists R.E)(a)$?



Approach:

1. Add negation of query:

   s4: $\forall R.\neg E(a)$

   to KB

2. Show unsatisfiability

based on slide from Pascal Hitzler

Tableaux example 2 Reasoning by Cases

KB:

s1: C(a)

s2: $C \sqsubseteq \exists R.D$

s3: $D \sqsubseteq E \sqcup F$

s4: $F \sqsubseteq E$

Query: Does KB entail

$(\exists R.E)(a)$?

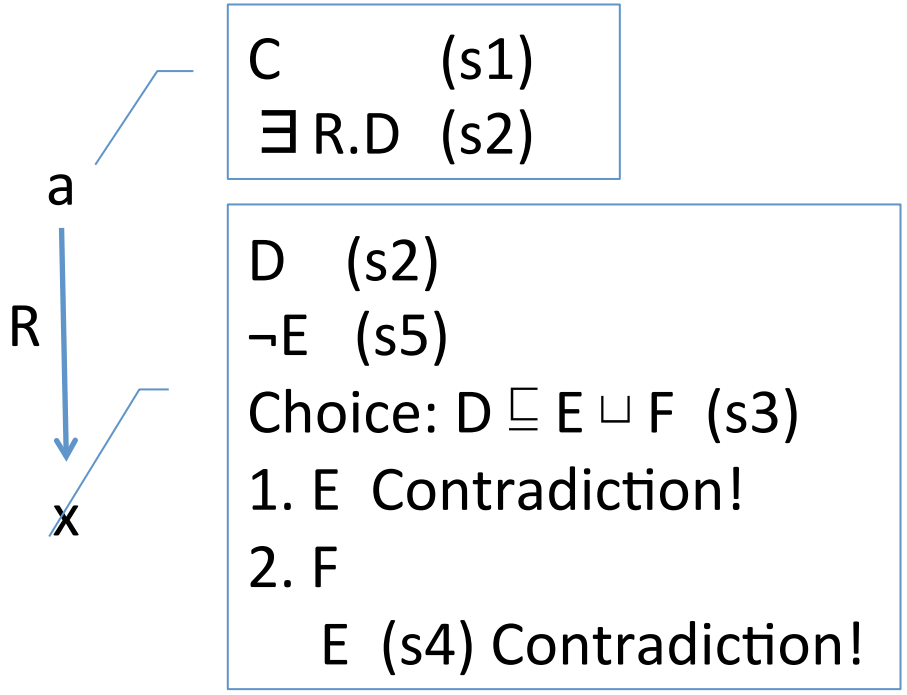
Approach:

1. Add negation of query:

   s5: $\forall R.\neg E(a)$

   to KB

2. Show unsatisfiability



based on slide from Pascal Hitzler

Example

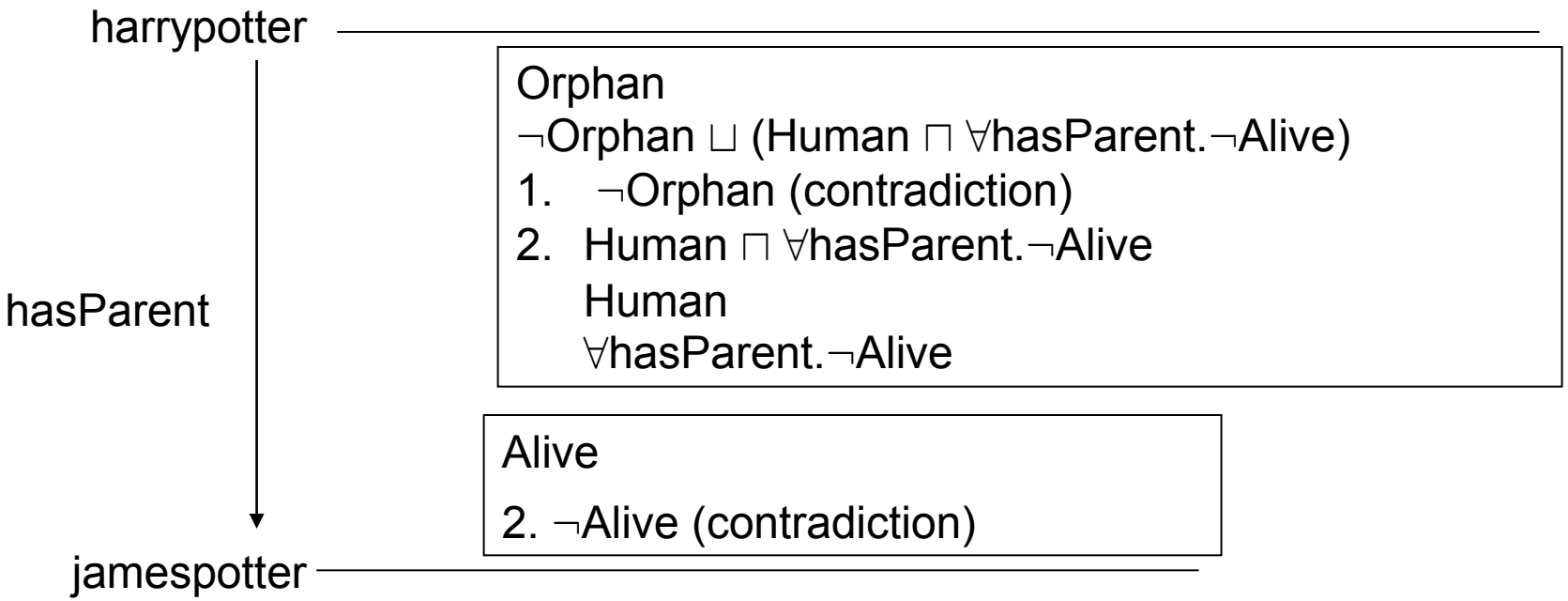
¬Alive(jamespotter)
i.e. add: Alive(jamespotter)
and search for contradiction

¬Human ⊔ ∃hasParent.Human

¬Orphan ⊔ (Human ⊓ ∀hasParent.¬Alive)

Orphan(harrypotter)

hasParent(harrypotter,jamespotter)



Termination

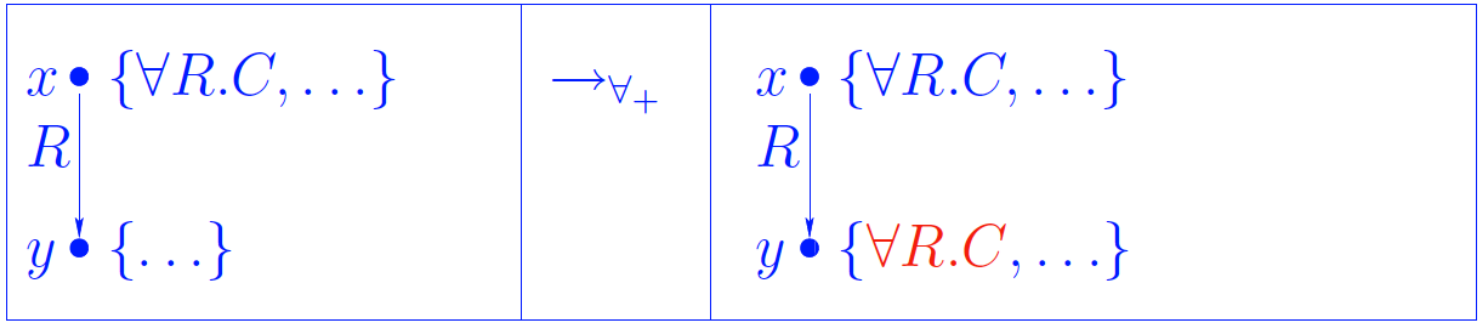
- New blocking rules to ensure termination
- See Ian Horrocks slides

Tableaux Rules for $\mathcal{ALC}$

$x : \{C_1 \sqcap C_2, \ldots\}$	$\rightarrow_{\sqcap}$	$x : \{C_1 \sqcap C_2, C_1, C_2, \ldots\}$
$x : \{C_1 \sqcup C_2, \ldots\}$	$\rightarrow_{\sqcup}$	$x : \{C_1 \sqcap C_2, C, \ldots\}$ for $C \in \{C_1, C_2\}$
$x : \{\exists R.C, \ldots\}$	$\rightarrow_{\exists}$	$x : \{\exists R.C, \ldots\}$ R $y : \{C\}$
$x : \{\forall R.C, \ldots\}$ R $y : \{\ldots\}$	$\rightarrow_{\forall}$	$x : \{\forall R.C, \ldots\}$ R $y : \{C, \ldots\}$

[slide by Ian Horrocks]

Tableaux Rule for Transitive Roles



Where R is a transitive role (i.e., $(R^{\mathcal{I}})^+ = R^{\mathcal{I}}$)

- No longer naturally terminating (e.g., if $C = \exists R.\top$)
- Need blocking
  - Simple blocking suffices for $\mathcal{ALC}$ plus transitive roles
  - I.e., do not expand node label if ancestor has superset label
  - More expressive logics (e.g., with inverse roles) need more sophisticated blocking strategies

[slide by Ian Horrocks]

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

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

[slide by Ian Horrocks]

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

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

[slide by Ian Horrocks]

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

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

[slide by Ian Horrocks]

Tableaux Algorithm — Example

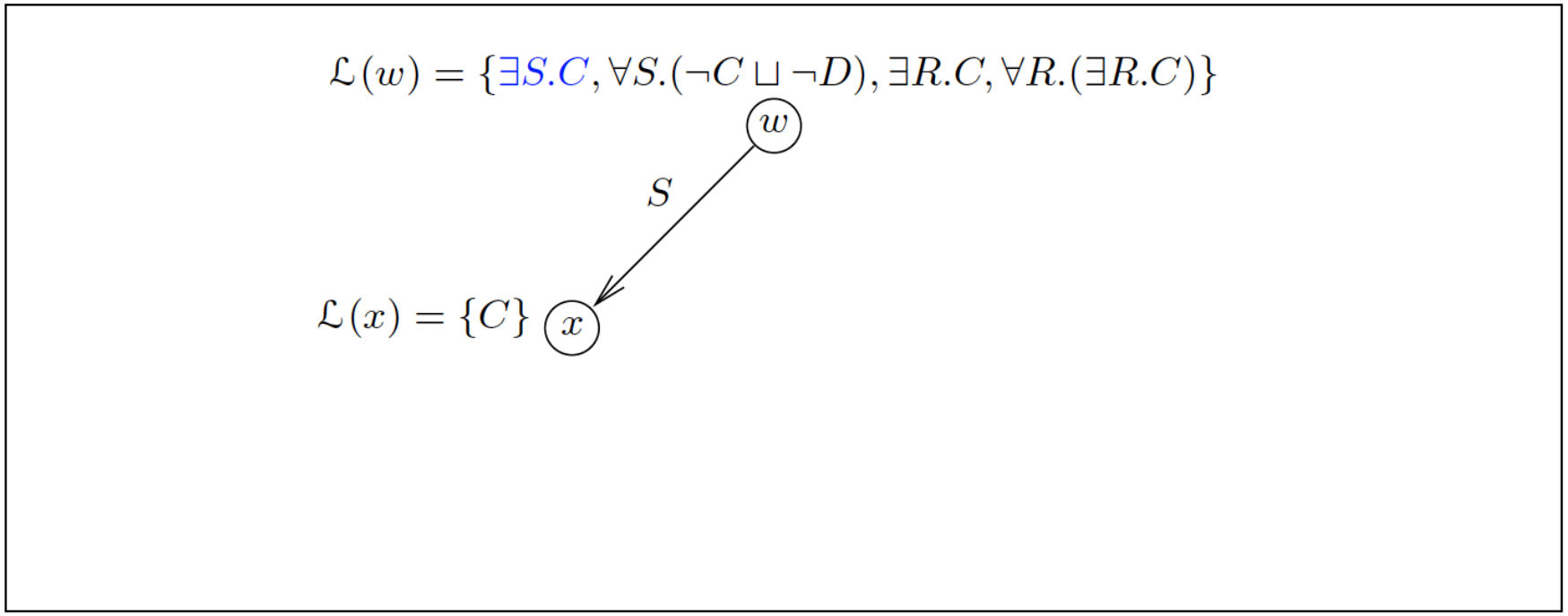
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role

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

[slide by Ian Horrocks]

Tableaux Algorithm — Example

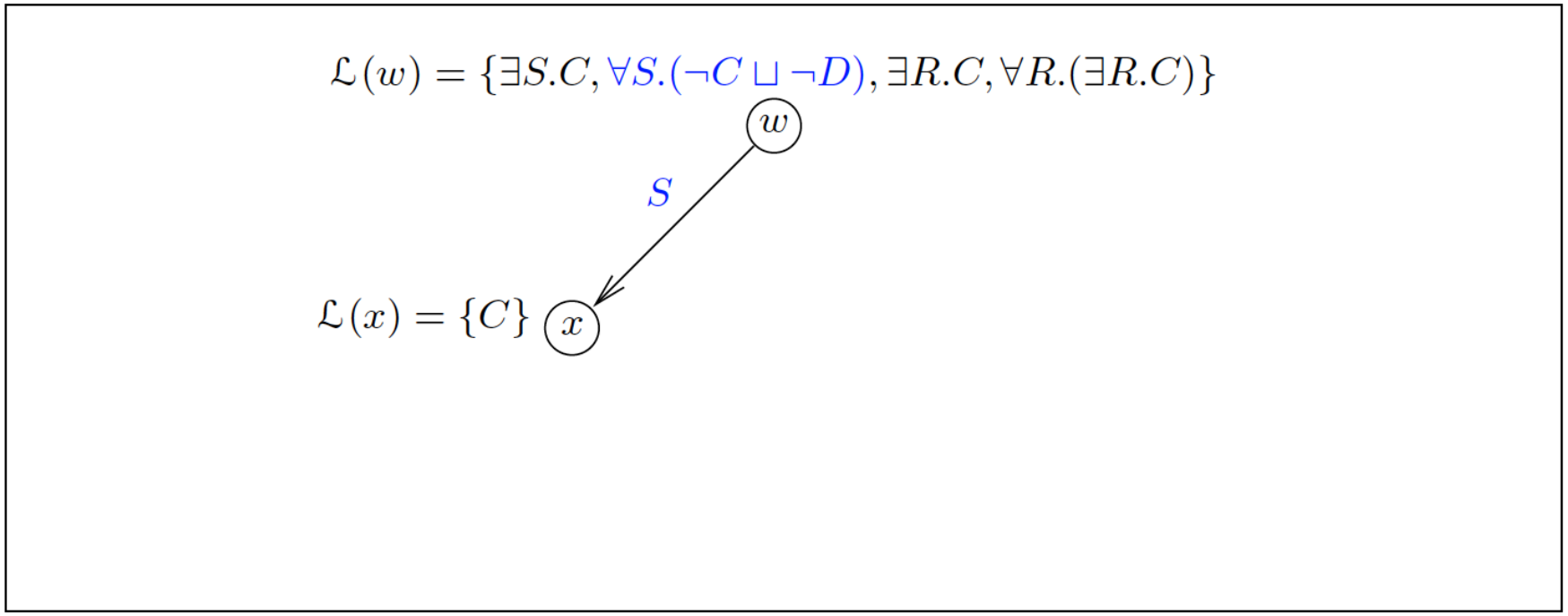
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

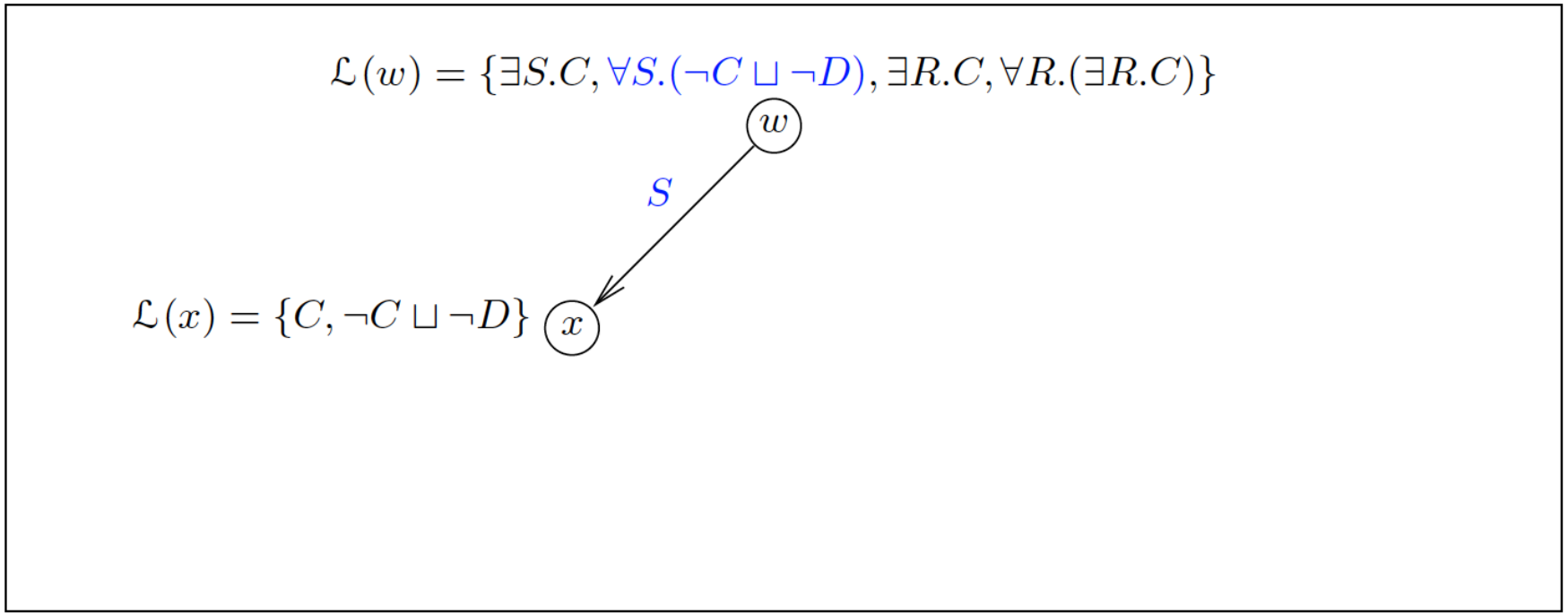
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

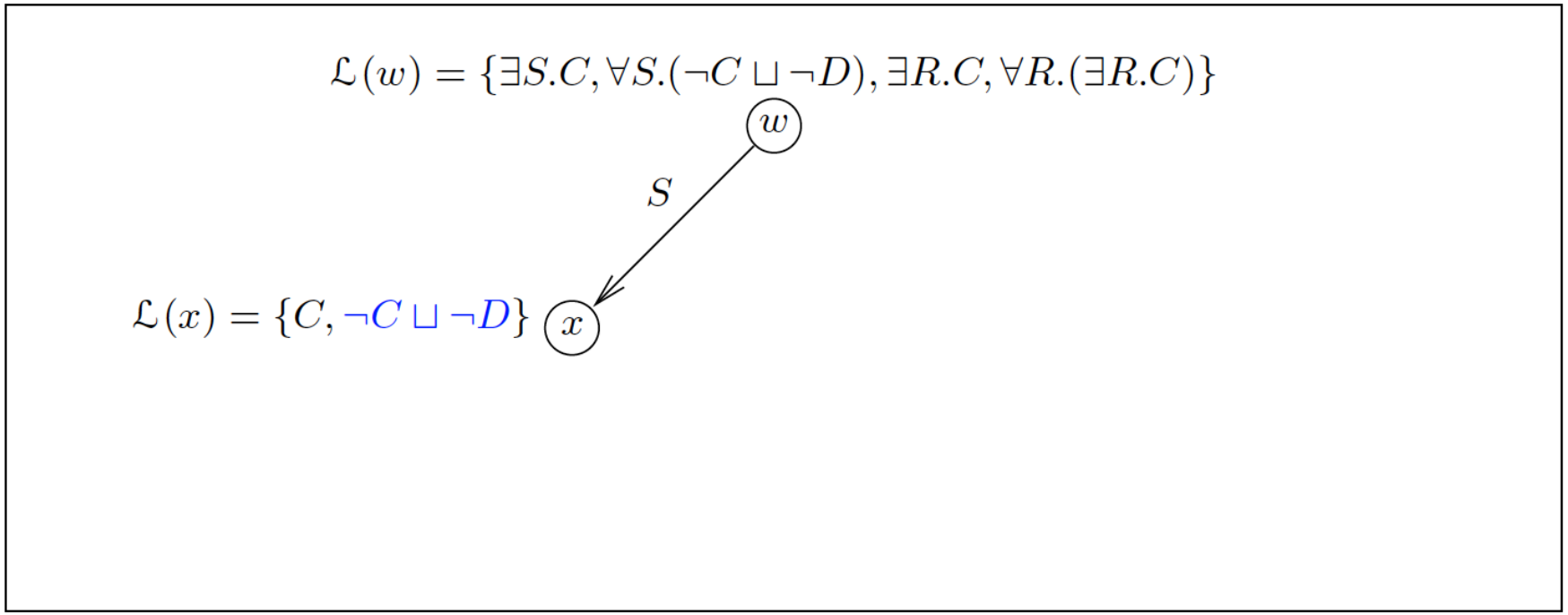
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

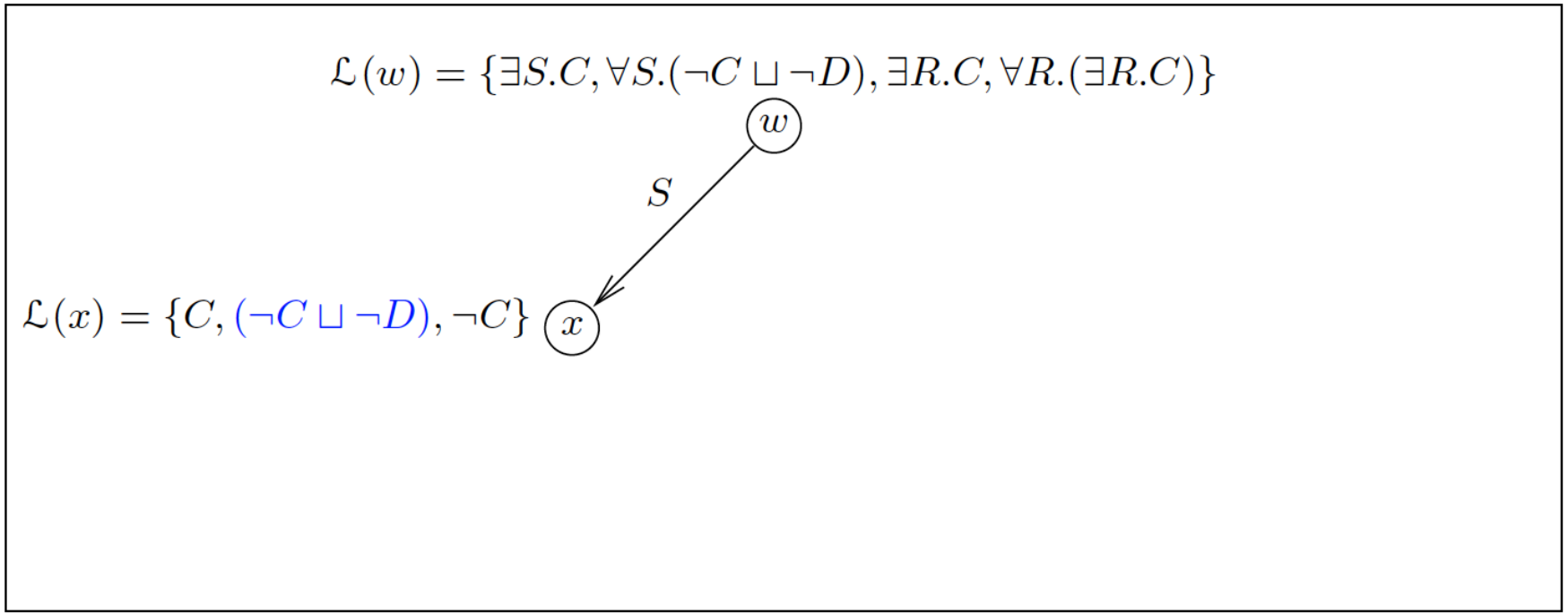
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

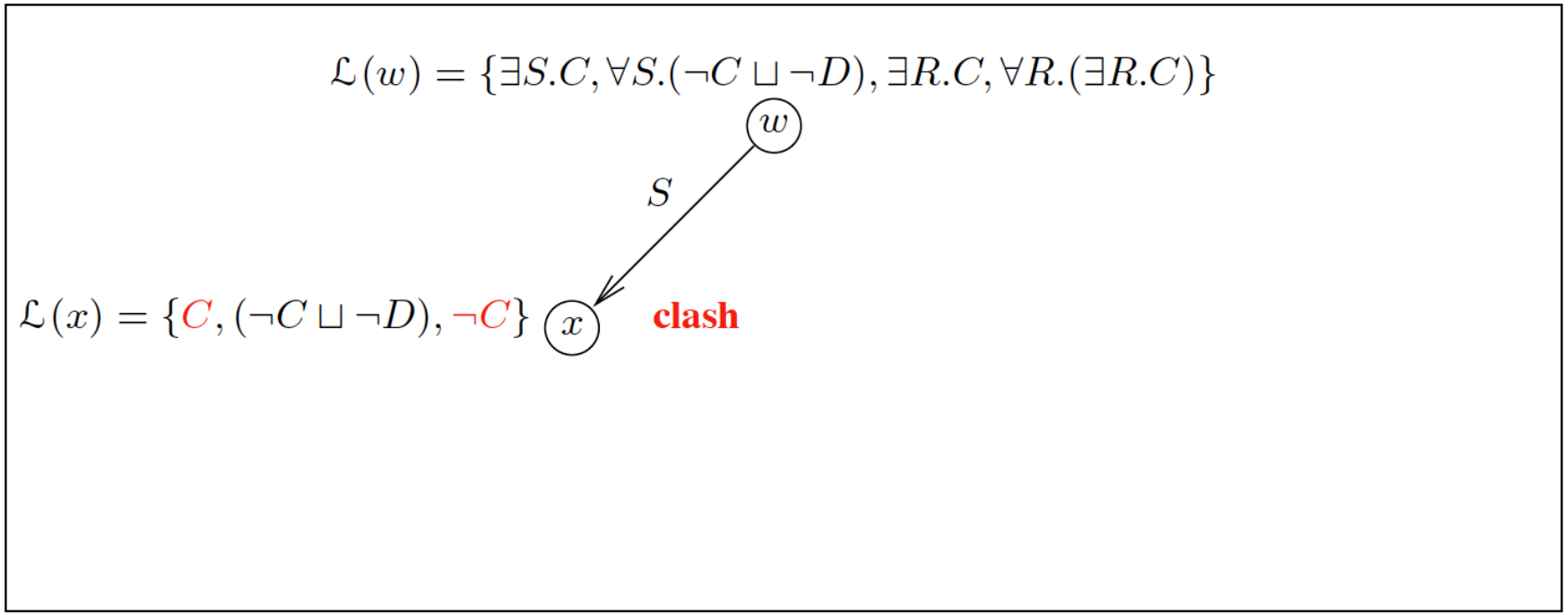
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

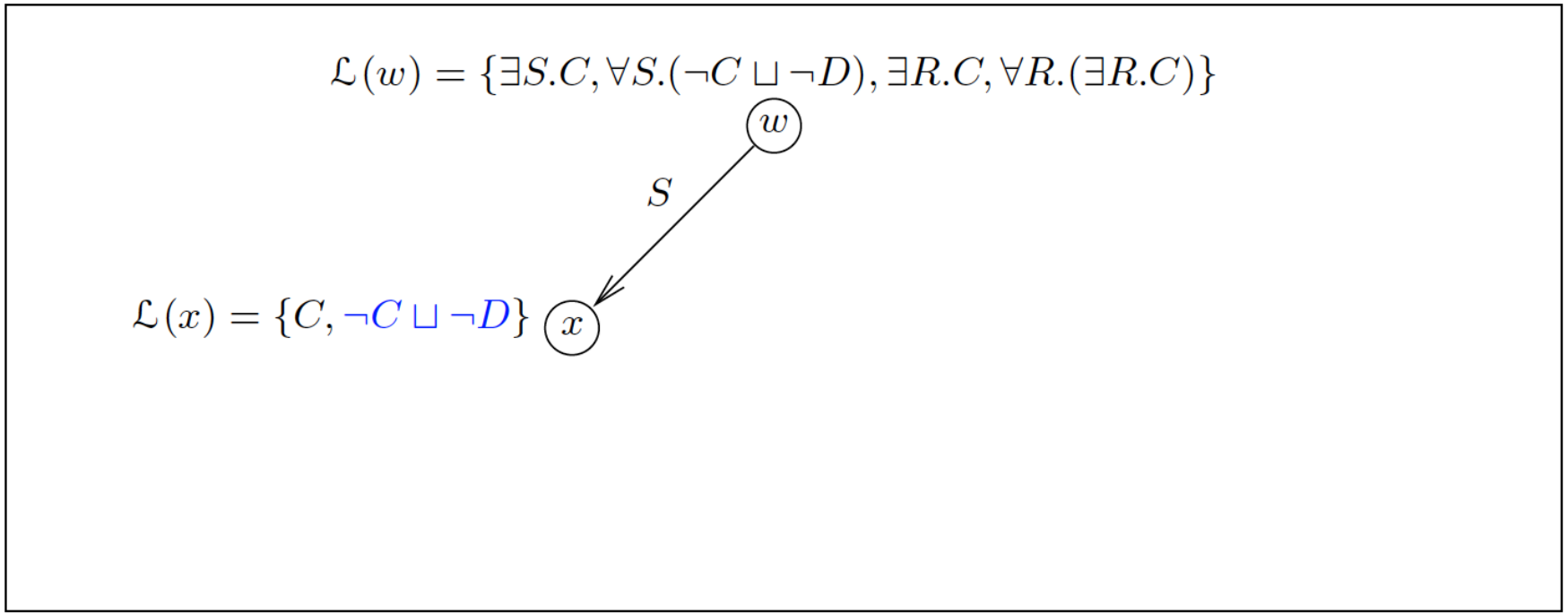
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

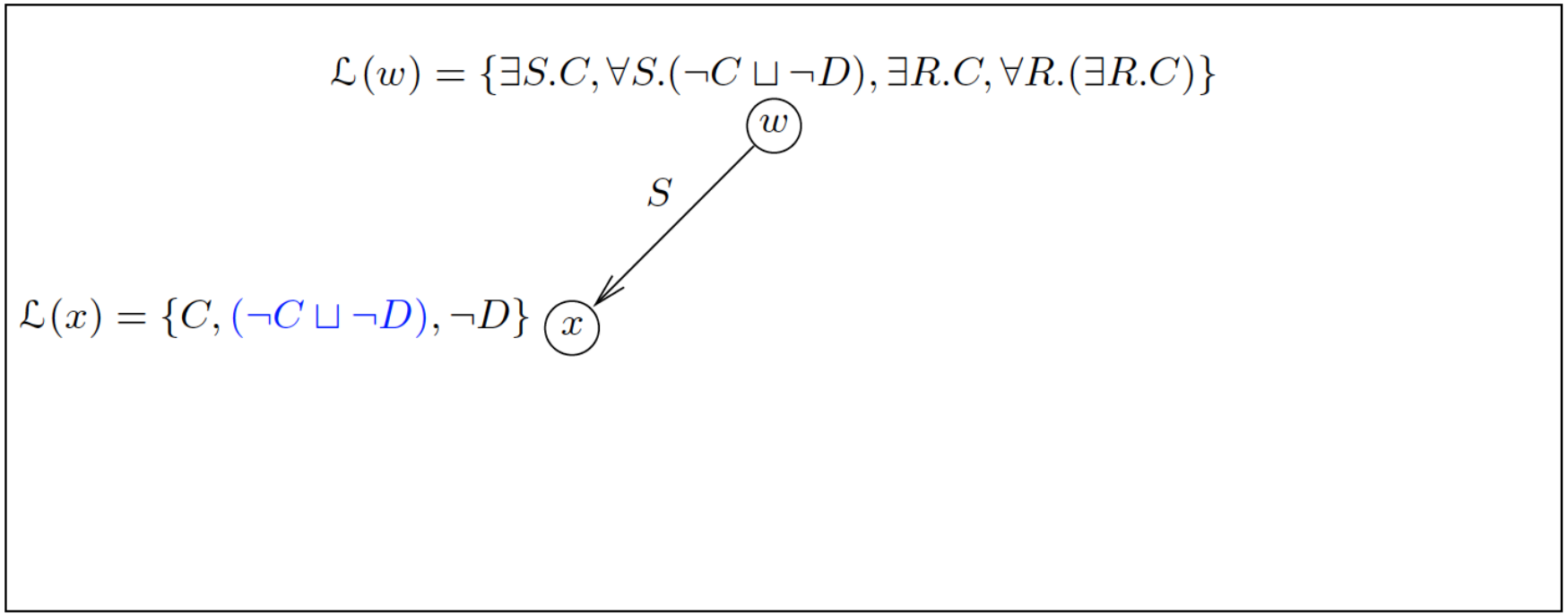
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

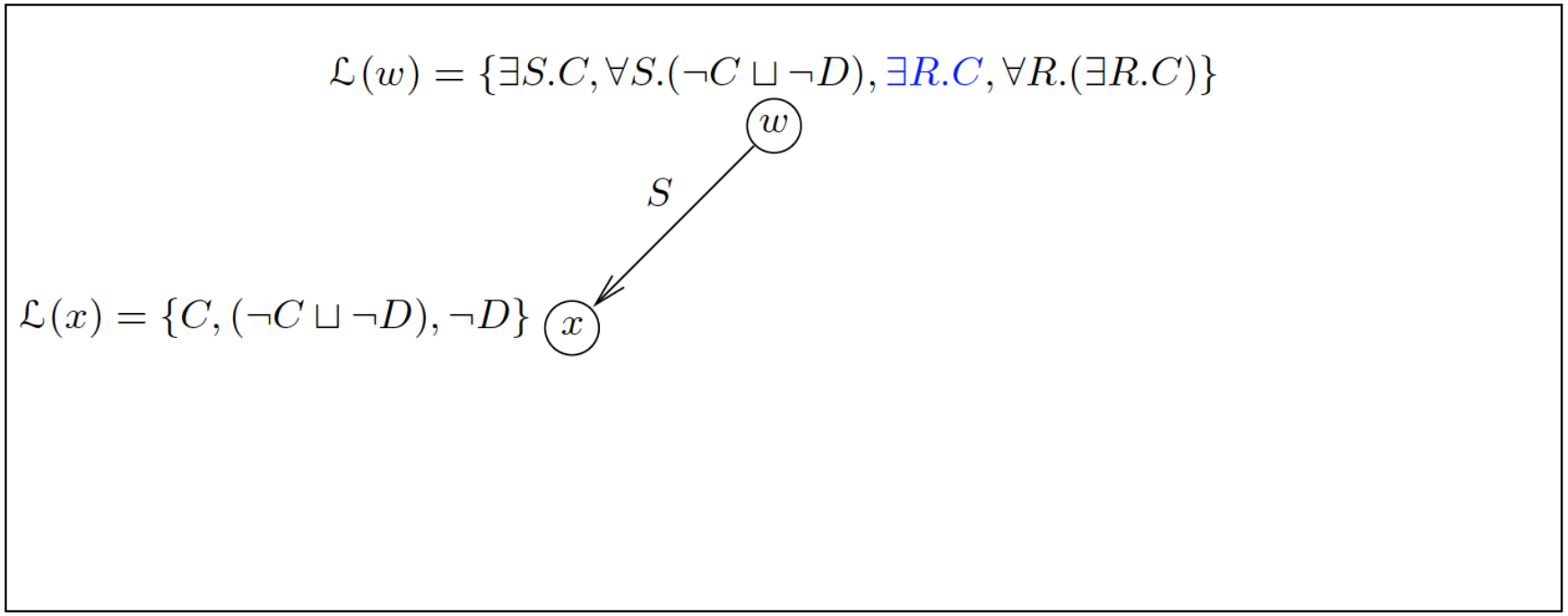
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

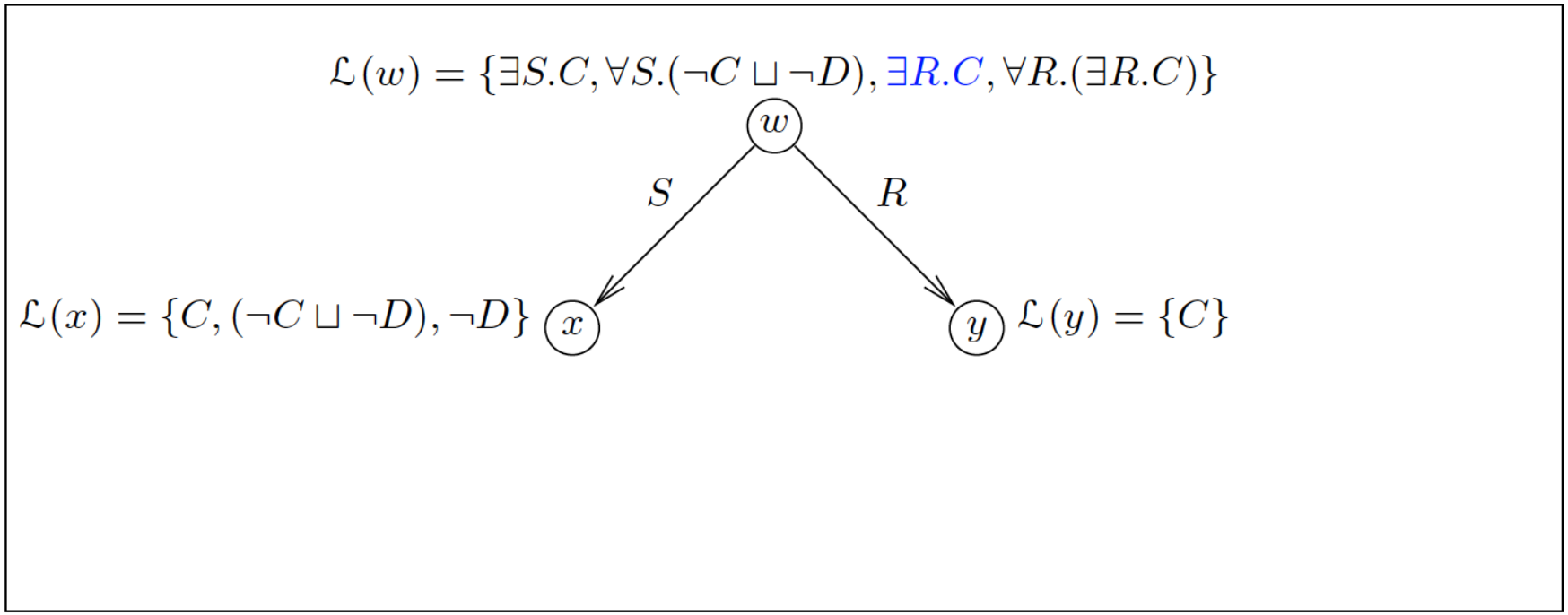
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

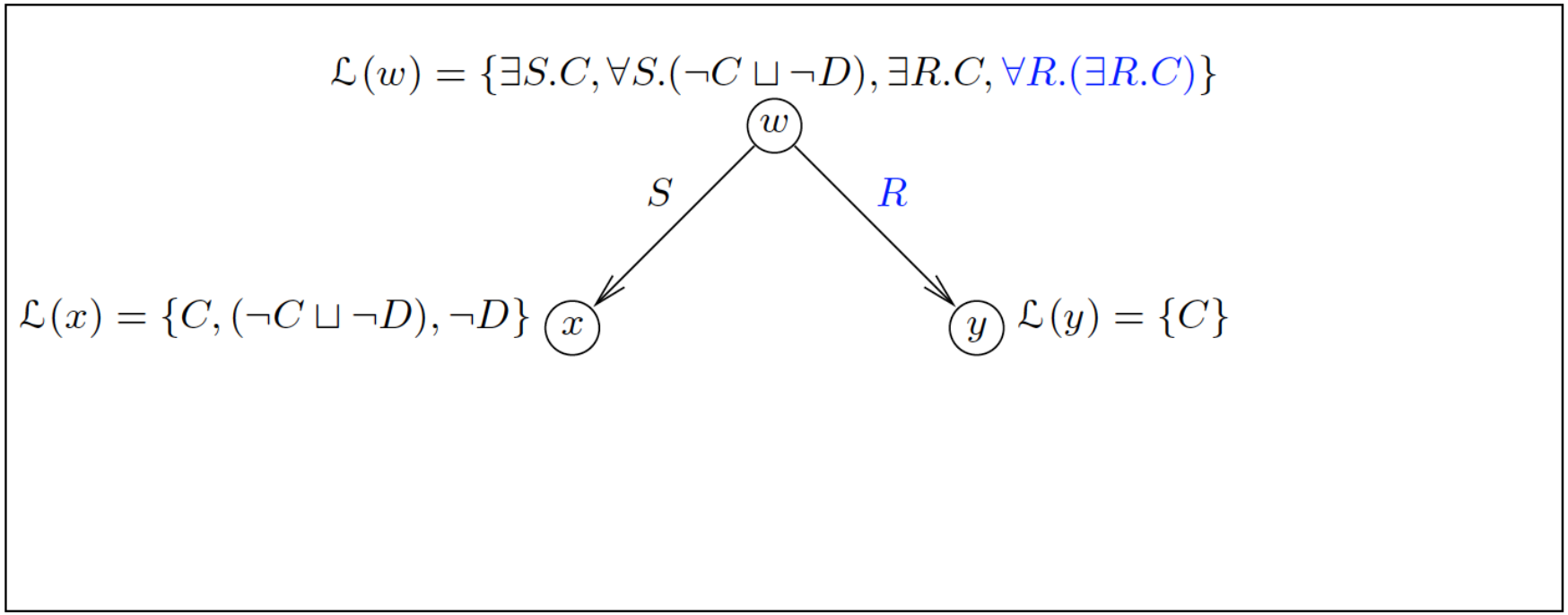
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

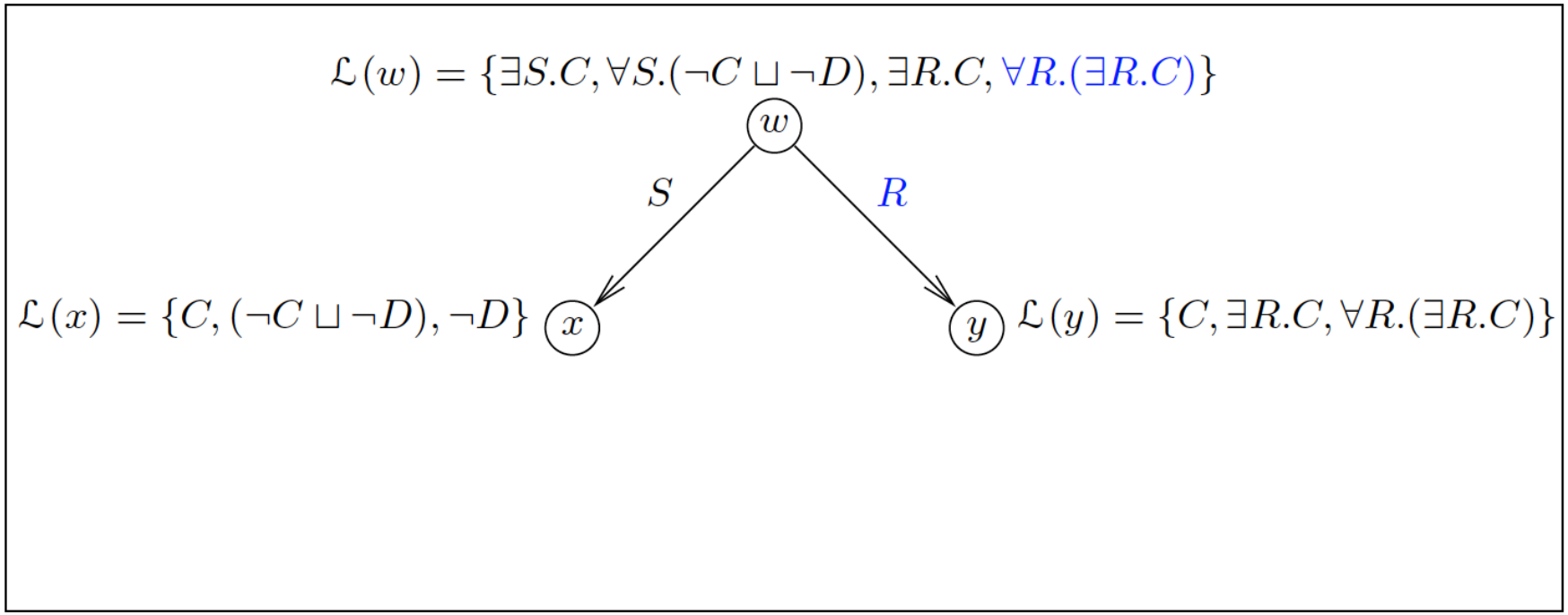
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

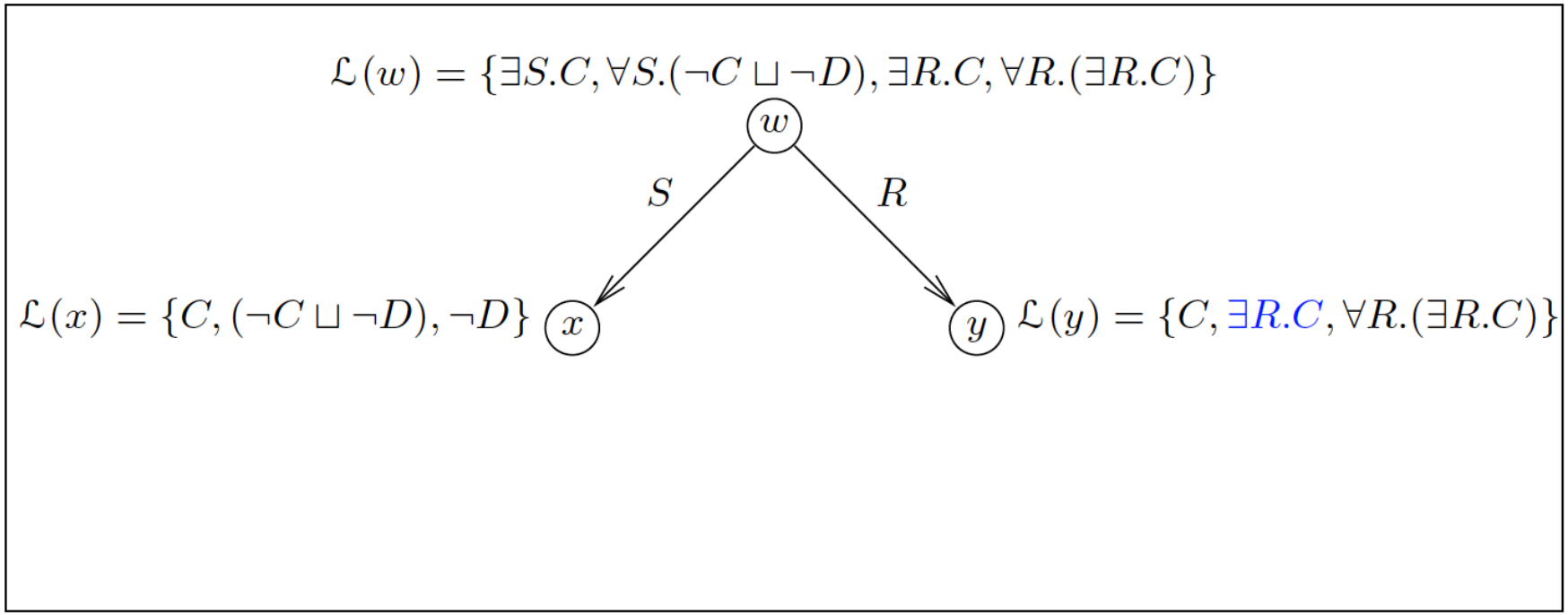
Tableaux Algorithm — Example

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

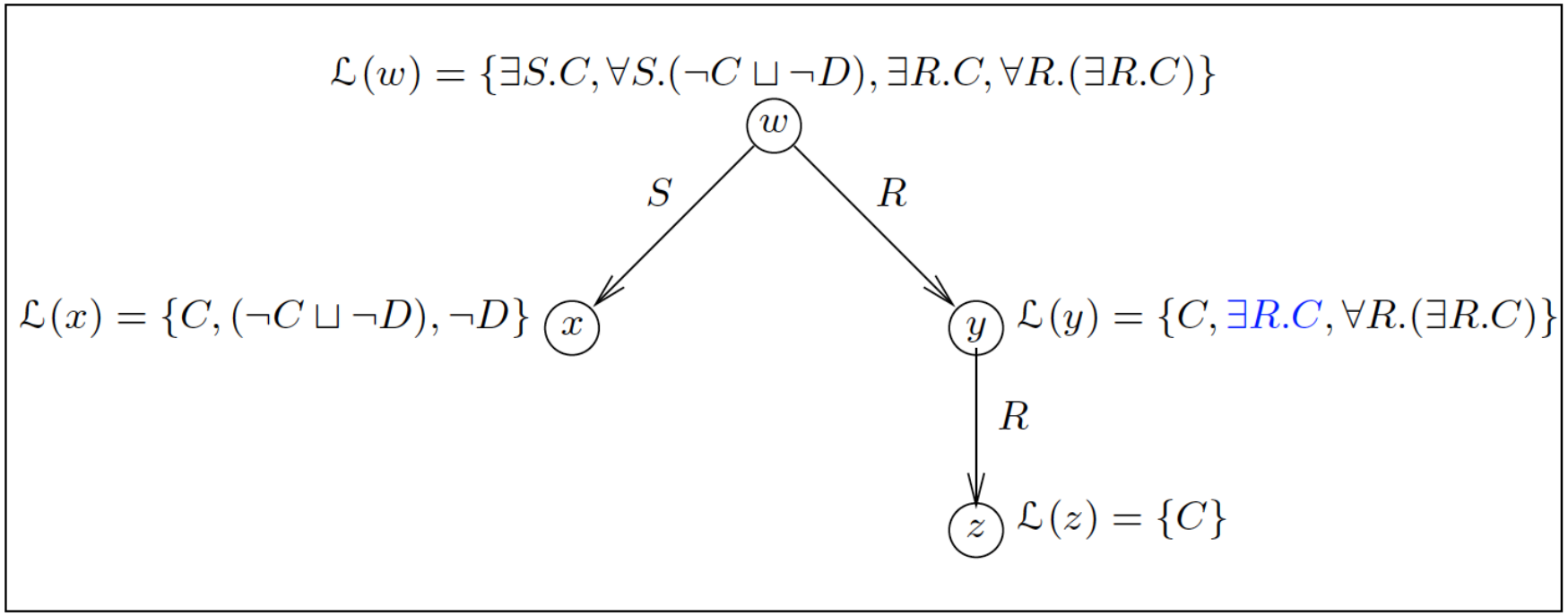
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

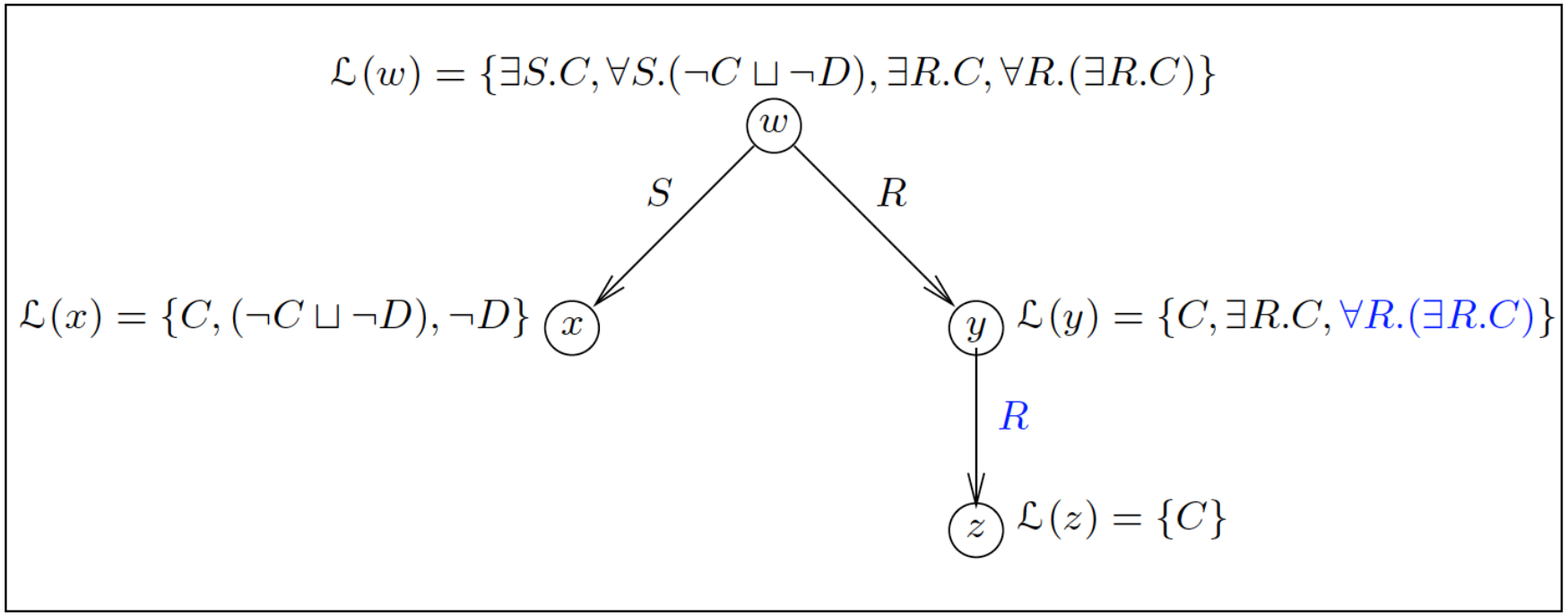
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

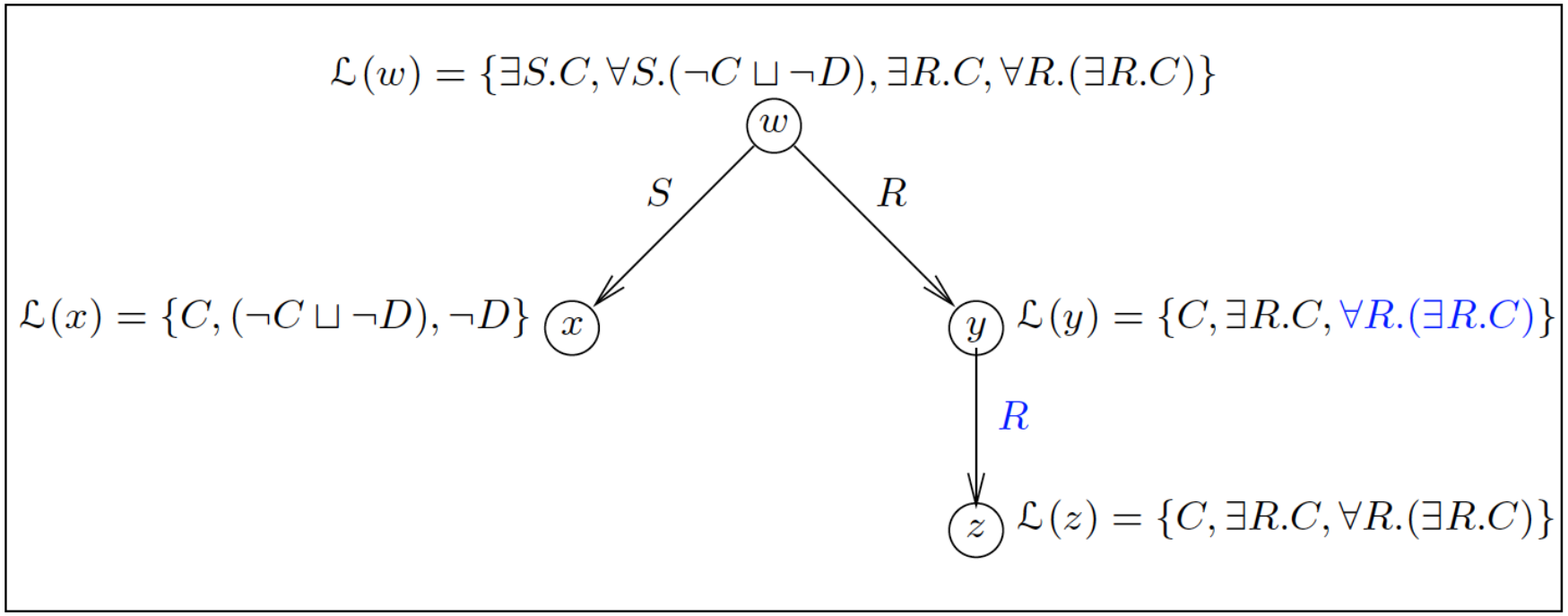
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

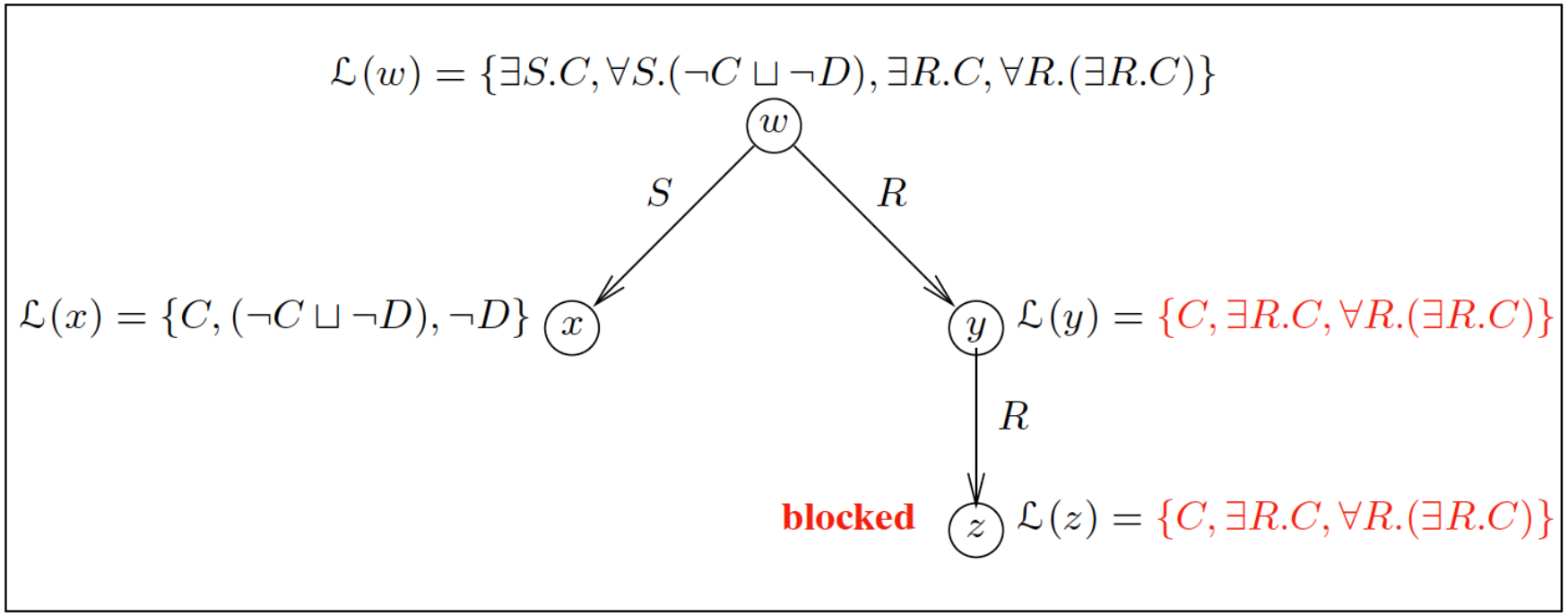
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

Tableaux Algorithm — Example

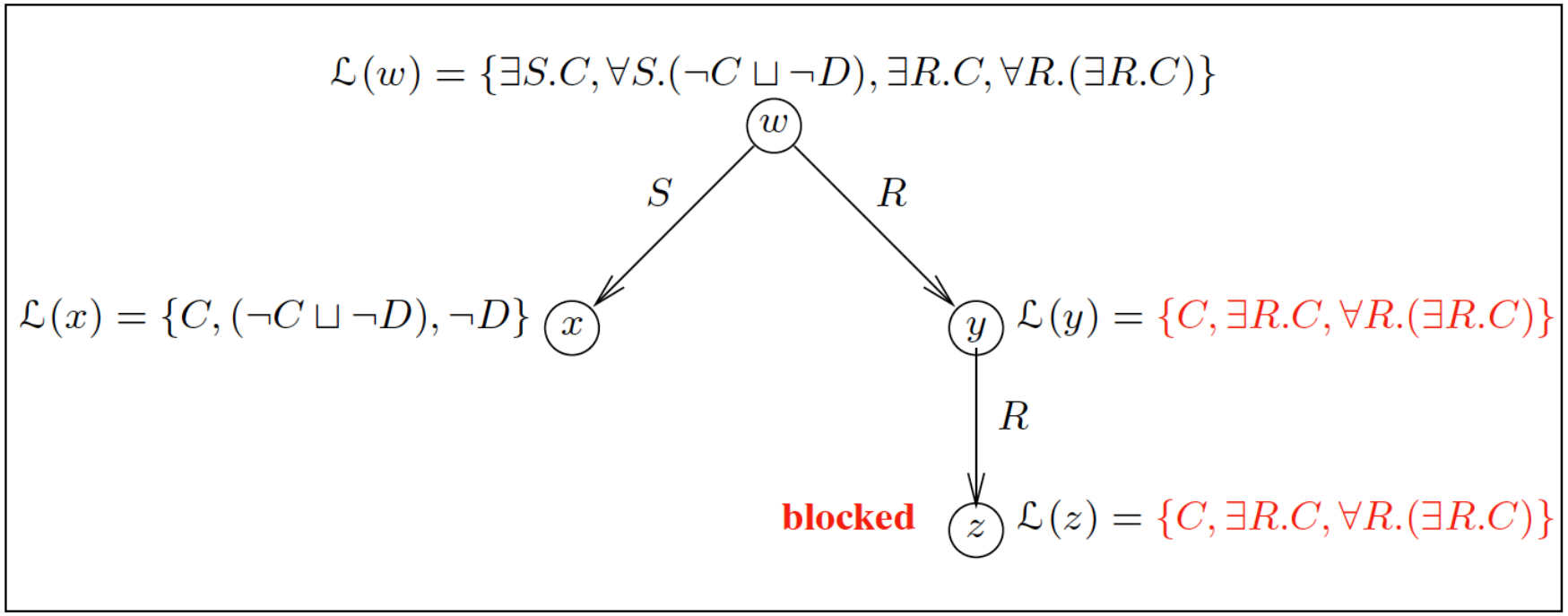
Test satisfiability of $\exists S.C \sqcap \forall S.(\neg C \sqcup \neg D) \sqcap \exists R.C \sqcap \forall R.(\exists R.C)\}$ where R is a **transitive** role



[slide by Ian Horrocks]

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

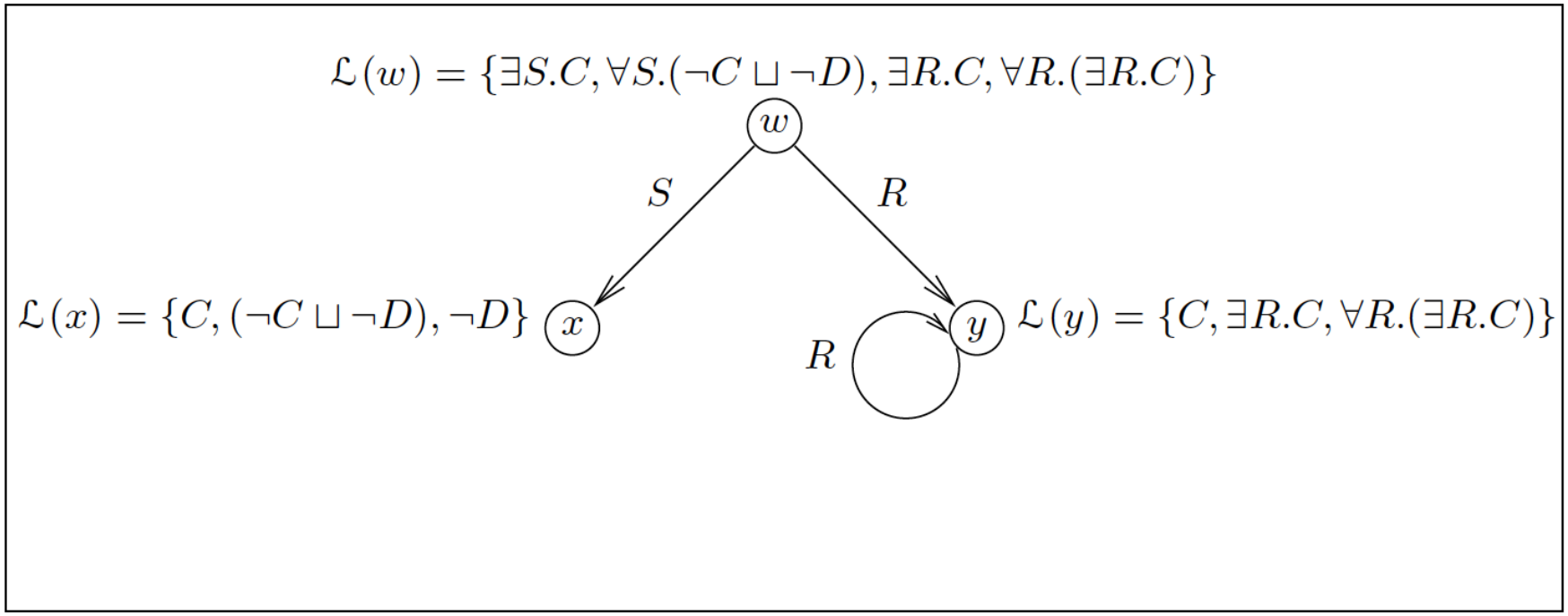


Concept is **satisfiable**: $\mathbf{T}$ corresponds to **model**

[slide by Ian Horrocks]

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**

[slide by Ian Horrocks]

More Advanced Techniques

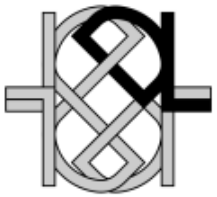
Satisfiability w.r.t. a Terminology

- For each axiom $C \sqsubseteq D \in \mathcal{T}$, add $\neg C \sqcup D$ to **every** node label

More expressive DLs

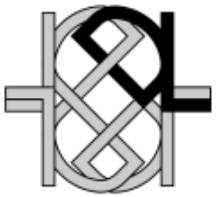
- Basic technique can be extended to deal with
  - Role inclusion axioms (role hierarchy)
  - Number restrictions
  - Inverse roles
  - Concrete domains and datatypes
  - Aboxes
  - etc.
- Extend **expansion rules** and use more sophisticated **blocking** strategy
- **Forest** instead of Tree (for Aboxes)
  - Root nodes correspond to individuals in Abox

[slide by Ian Horrocks]



Complexity of reasoning in Description Logics

Note: the information here is (always) incomplete and updated often



Base description logic: $\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplements

$\mathcal{ALC}$ ::= $\bot$ | A | $\neg C$ | $C \wedge D$ | $C \vee D$ | $\exists R.C$ | $\forall R.C$

ALC

Concept constructors:

- ☐ $\mathcal{F}$ – functionality[2]: ($\leq 1\ R$)
- ☐ $\mathcal{N}$ – (unqualified) number restrictions: ($\geq n\ R$), ($\leq n\ R$)
- ☐ $\mathcal{Q}$ – qualified number restrictions: ($\geq n\ R.C$), ($\leq n\ R.C$)
- ☐ $\mathcal{O}$ – nominals: $\{a\}$ or $\{a_1,\ldots,a_n\}$ ("one-of" constructor)

- ☐ μ – least fixpoint operator: $\mu X.C$
- ☐ $R \sqsubseteq S$ – role-value-maps
- ☐ $f = g$ – agreement of functional role chains ("same-as")

Role constructors:

trans reg

- ☐ $\mathcal{I}$ – role inverses: R^-

- ☐ $\cap$ – role intersection[3]: $R \cap S$
- ☐ $\cup$ – role union: $R \cup S$
- ☐ $\neg$ – role complement: full
- ☐ o – role chain (composition): $R \circ S$
- ☐ * – reflexive-transitive closure[4]: R^*
- ☐ id – concept identity: $id(C)$

Forbid complex roles[5] in number restrictions[6]

TBox options:

- ◉ Empty TBox
- ○ Acyclic TBox ($A \equiv C$, A is a concept name; no cycles)
- ○ General TBox ($C \sqsubseteq D$ for arbitrary concepts C and D)

Role axioms (RBox):

OWL-Lite
OWL-DL
OWL 1.1

- ☐ $\mathcal{S}$ – Role transitivity: Trans(R)
- ☐ $\mathcal{H}$ – Role hierarchy: $R \sqsubseteq S$
- ☐ $\mathcal{R}$ – Complex role inclusions: $R \circ S \sqsubseteq R$, $R \circ S \sqsubseteq S$
- ☐ $\mathcal{s}$ – some additional features

Reset

You have selected the Description Logic: $\mathcal{ALC}$

Complexity of reasoning problems[7]		
Reasoning problem	**Complexity[8]**	**Comments and references**
Concept satisfiability	**PSpace-complete**	• Hardness for $\mathcal{ALC}$: see [73]. • Upper bound for $\mathcal{ALCQ}$: see [77, Theorem 4.6].
ABox consistency	**PSpace-complete**	• Hardness follows from that for concept satisfiability. • Upper bound for $\mathcal{ALCQO}$: see [7, Appendix A].
Important properties of the description logic		
Finite model property	**Yes**	$\mathcal{ALC}$ is a notational variant of the multi-modal logic $\mathbf{K_m}$ (cf. [70]), for which the finite model property can be found in [2, Sect. 2.3].

Complexity of reasoning in Description Logics

Note: the information here is (always) incomplete and updated often

Base description logic: $\mathcal{A}$ttributive $\mathcal{L}$anguage with $\mathcal{C}$omplements
$\mathcal{ALC}$ ::= $\bot$ | $\top$ | A | $\neg C$ | $C \sqcap D$ | $C \sqcup D$ | $\exists R.C$ | $\forall R.C$

Concept constructors:

- ☑ $\mathcal{F}$ – functionality[2]: $(\leq 1\ R)$
- ☐ $\mathcal{N}$ – (unqualified) number restrictions: $(\geq n\ R)$, $(\leq n\ R)$
- ☑ $\mathcal{Q}$ – qualified number restrictions: $(\geq n\ R.C)$, $(\leq n\ R.C)$
- ☑ $\mathcal{O}$ – nominals: $\{a\}$ or $\{a_1, \ldots, a_n\}$ ("one-of")
- ☐ μ – least fixpoint operator: $\mu X.C$

Forbid complex roles[5] in number restrictions[6]

Role constructors:

trans reg

- ☑ $\mathcal{I}$ – role inverse: R^-
- ☐ ∩ – role intersection[3]: $R \cap S$
- ☐ ∪ – role union: $R \cup S$
- ☐ ¬ – role complement: $\neg R$ full
- ☐ o – role chain (composition): $R \circ S$
- ☐ * – reflexive-transitive closure[4]: R^*
- ☐ *id* – concept identity: $id(C)$

SHOIQ

TBox (concept axioms) is *internalizable* in extensions of $\mathcal{ALCIO}$, see [82, Lemma 4.12], [61, p.3]

- ◉ empty TBox
- ○ acyclic TBox ($A \equiv C$, A is a concept name; no cycles)
- ○ general TBox ($C \sqsubseteq D$, for arbitrary concepts C and D)

RBox (role axioms):

OWL-Lite OWL-DL OWL 1.1

- ☑ $\mathcal{S}$ – role transitivity: $\mathrm{Tr}(R)$
- ☑ $\mathcal{H}$ – role hierarchy: $R \sqsubseteq S$
- ☐ $\mathcal{R}$ – complex role inclusions: $R \circ S \sqsubseteq R$, $R \circ S \sqsubseteq S$
- ☐ $\mathcal{s}$ – some additional features (click to see them)

Reset You have selected a Description Logic: $\mathcal{SHOIQ}$

Complexity[7] of reasoning problems[8]

Concept satisfiability	**NExpTime-complete**	• Hardness of even $\mathcal{ALCFIO}$ is proved in [82, Corollary 4.13]. • A different proof of the NExpTime-hardness for $\mathcal{ALCFIO}$ is given in [61] (even with 1 nominal, and inverse roles not used in number restrictions). • Upper bound for $\mathcal{SHOIQ}$ is proved in [12, Corollary 6.31] with numbers coded in unary (for binary coding, the upper bound remains an open problem for all logics in between $\mathcal{ALCNIO}$ and $\mathcal{SHOIQ}$). • A tableaux algorithm for $\mathcal{SHOIQ}$ is presented in [51]. • **Important:** in number restrictions, only *simple* roles (i.e. which are neither transitive nor have a transitive subroles) are allowed; otherwise we gain undecidability even in $\mathcal{SHN}$; see [54]. cases, one can use transitive roles in So the above notion of a *simple* role

http://www.cs.man.ac.uk/~ezolin/dl/

Resources for OWL and DL

- Description Logic Handbook, Cambridge University Press
  - http://books.cambridge.org/0521781760.htm
- Description Logic: http://dl.kr.org/
  - complexity: http://www.cs.man.ac.uk/~ezolin/dl
- Web Ontology Language (OWL):
  - http://www.w3.org/TR/owl2-overview
- Reasoners:
  - Pellet (open source): http://pellet.owldl.com/
  - FaCT++ (open source): http://owl.man.ac.uk/factplusplus/
  - Racer (comercial): http://www.racer-systems.com/
  - (Loom and Powerloom: http://www.isi.edu/isd/LOOM/)
- Ontology Editors:
  - Protégé: http://protege.stanford.edu/
- Ian Horrocks has great slides on description logics and OWL:
  - http://web.comlab.ox.ac.uk/oucl/work/ian.horrocks/