



University of
Southampton

RDF Schema and Description Logics

COMP6256 Knowledge Graphs for AI Systems

Dr Nicholas Gibbins – nmg@ecs.soton.ac.uk

Using RDF to define RDFS

RDFS is a simple ontology language for use with RDF

RDFS is an RDF vocabulary which contains:

- Classes for defining classes and properties
- Properties for defining basic characteristics of classes and properties
 - Global property domains and ranges
- Some ancillary properties
 - Defined by, see also

Notes on RDF and RDFS namespaces

Most terms in RDF Schema are defined as part of the RDFS namespace

- <http://www.w3.org/2000/01/rdf-schema#> , abbreviated here as `rdfs:`

Two terms are defined as part of the RDF namespace: `rdf:type` and `rdf:Property`

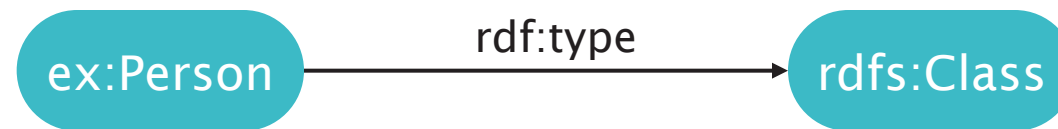
- <http://www.w3.org/1999/02/22-rdf-syntax-ns#> , abbreviated as `rdf:`

This is a historical accident, but can trip up the unwary

Be careful when using these terms in SPARQL queries!

RDF Schema class definitions

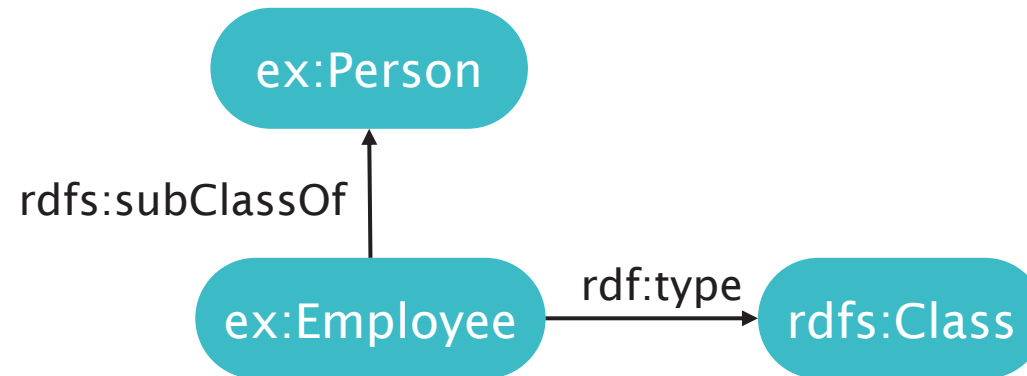
We wish to define the class Person:



```
ex:Person rdf:type rdfs:Class .
```

RDF Schema class definitions

Employee is a subclass of Person

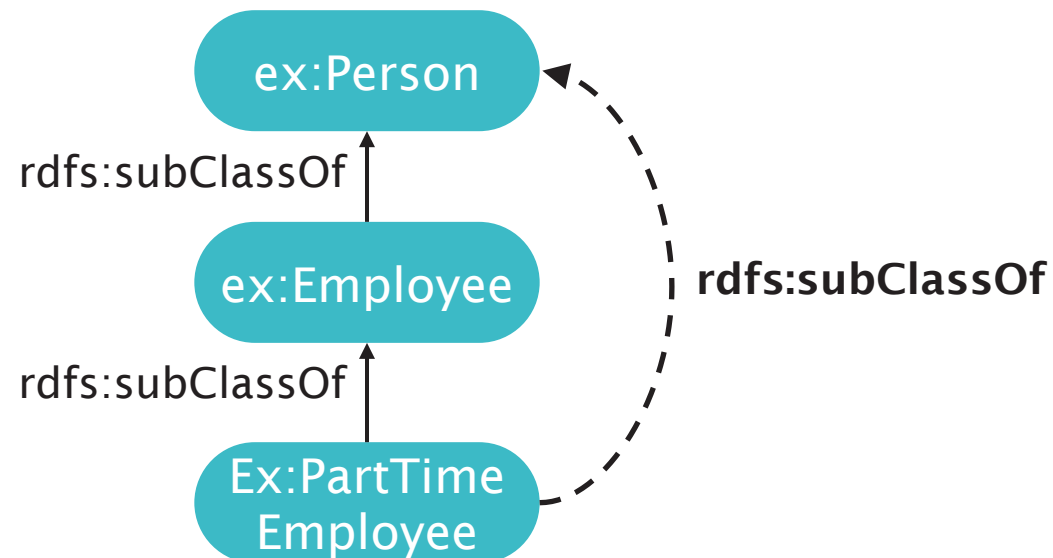


```
ex:Employee rdf:type rdfs:Class ;  
            rdfs:subClassOf ex:Person .
```

RDF Schema class semantics

rdfs:subClassOf is transitive:

(A rdfs:subClassOf B) and (B rdfs:subClassOf C) implies (A rdfs:subClassOf C)



RDF Schema class semantics

rdfs:subClassOf is reflexive

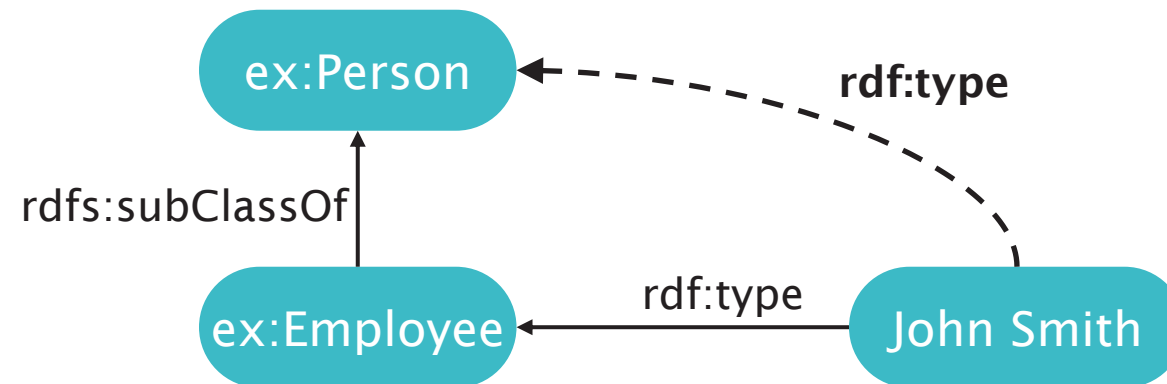
- All classes are subclasses of themselves



RDF Schema class semantics

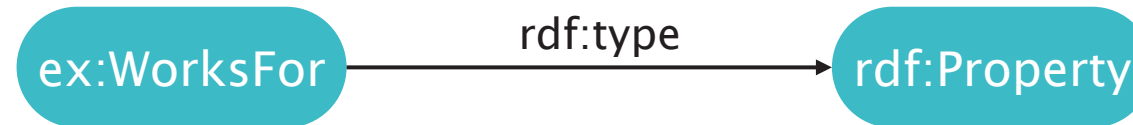
rdf:type distributes over rdfs:subClassOf:

(A rdfs:subClassOf B) and (C rdf:type A) implies (C rdf:type B)



RDF Schema property definitions

We wish to define the property worksFor:



```
ex:WorksFor rdf:type rdf:Property .
```

RDF Schema property definitions

Important difference between RDF and object-oriented programming languages

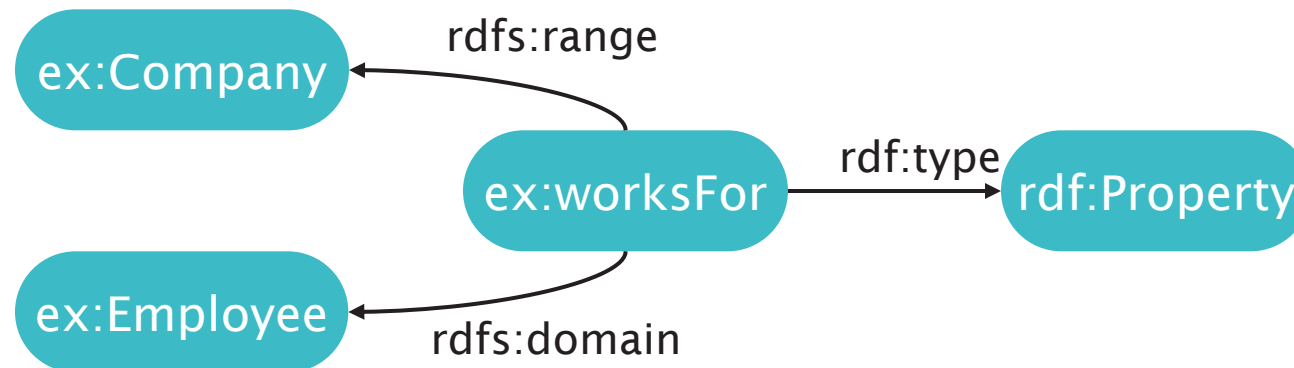
- OO languages define classes in terms of the properties they have
- RDF defines properties in terms of the classes whose instances they relate to each other

The *domain* of a property is the class that the property runs *from*

The *range* of a property is the class that a property runs *to*

RDF Schema property definitions

The property worksFor relates objects of class Employee to objects of class Company

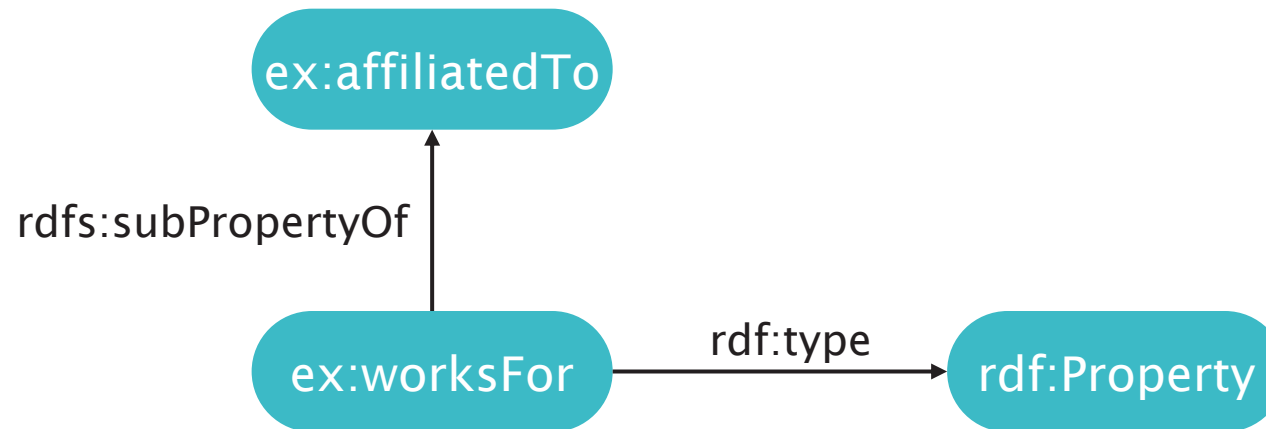


```
ex:worksFor rdf:type rdf:Property ;  
            rdfs:domain ex:Employee ;  
            rdfs:range ex:Company .
```

RDF Schema property definitions

Specialisation exists in properties as well as classes

- worksFor is a subproperty of affiliatedTo

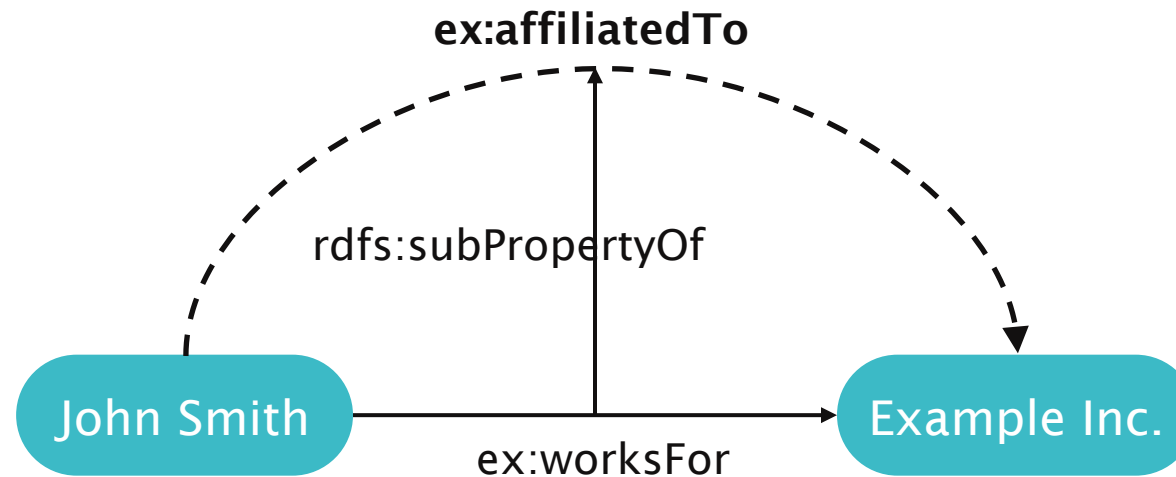


```
ex:worksFor rdf:type rdf:Property ;  
            rdfs:subPropertyOf ex:affiliatedTo
```

RDF Schema property semantics

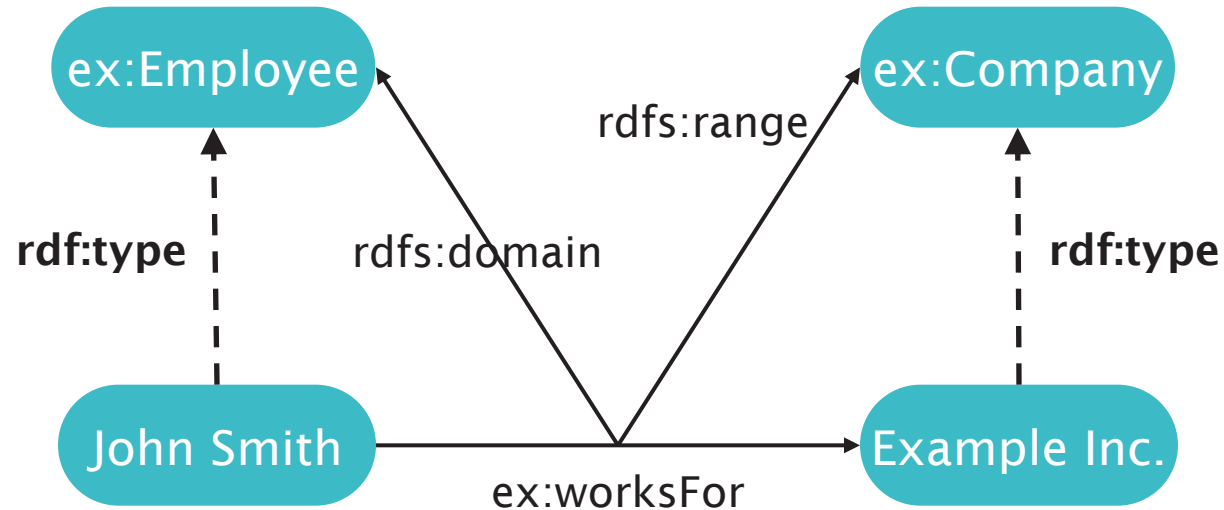
`rdfs:subPropertyOf` is transitive and reflexive

- Entailment of superproperties



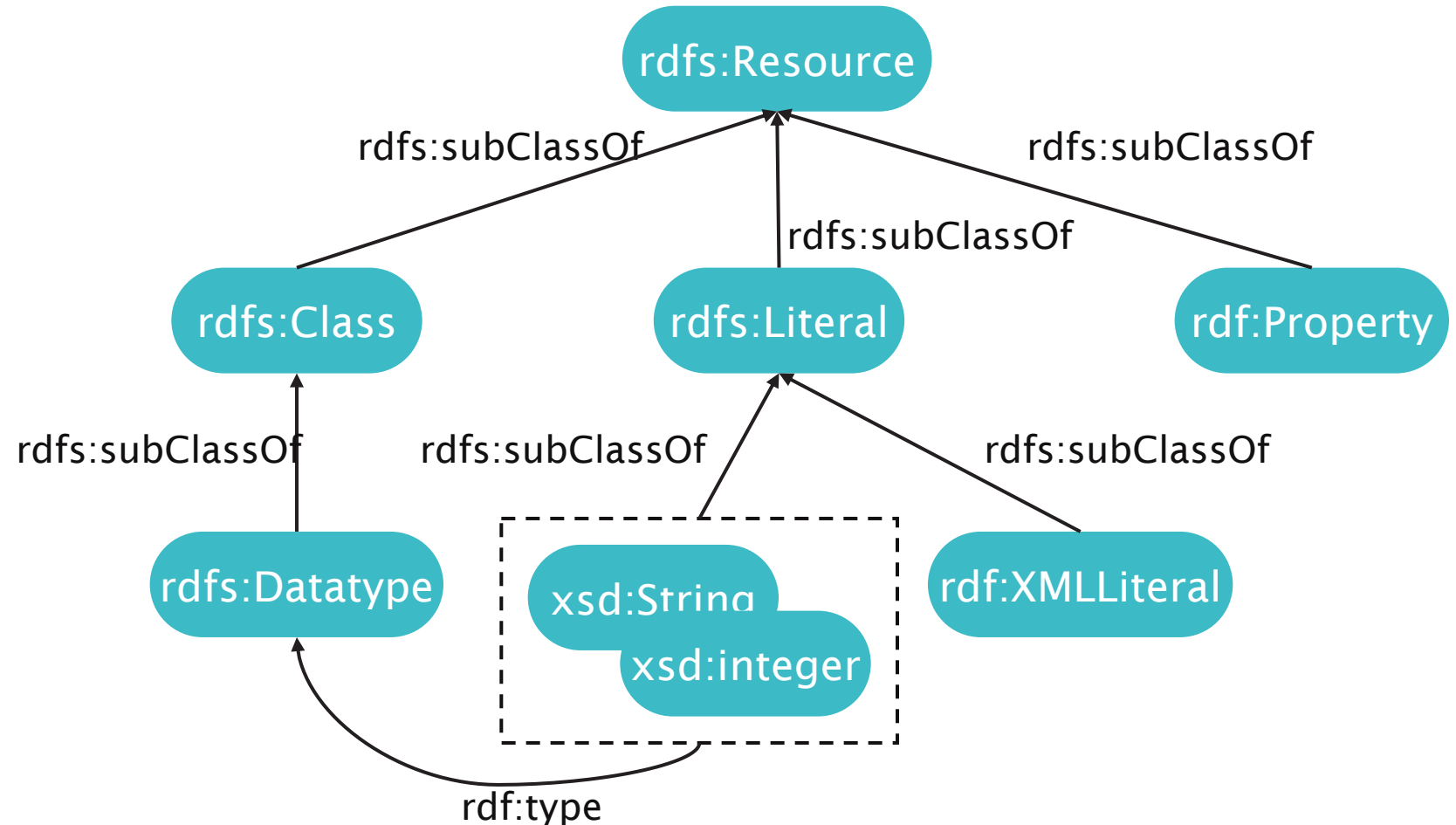
RDF Schema property semantics

Type entailments from range and domain constraints



RDF Schema predefined classes

- rdfs:Class
- rdf:Property
- rdfs:Resource
- rdfs:Literal
- rdfs:Datatype
- rdf:XMLLiteral



RDF Schema ancillary features

`rdfs:label` is used to give a human-readable name for a resource

```
<#person-01269> rdfs:label "John Smith" .
```

`rdfs:comment` is used to give a human-readable description for a resource

```
<#Employee> rdfs:comment "A person who works." .
```

`rdfs:seeAlso` is used to indicate a resource which can be retrieved to give more information about something

`rdfs:isDefinedBy` indicates a resource which is responsible for the definition of something (a subproperty of `rdfs:seeAlso`)

Description Logics

Why do we need Description Logics?

RDF Schema isn't sufficient for all tasks

- There are things you can't express
- There are things you can't infer

Description Logics

A *family* of knowledge representation formalisms

- A subset of first order predicate logic (FOPL)
- Decidable – trade-off of expressivity against algorithmic complexity
- Well understood – derived from work in the mid-80s to early 90s
- Model-theoretic formal semantics
- Simpler syntax than FOPL

Used as the foundation for the web ontology language OWL

This module assumes that you're familiar with FOPL.

If you need a refresher, the following resource is available:

- Johnsonbaugh, R. (2014) Discrete Mathematics, 7th ed. Chapter 1. (ebook via library)

Description Logics

Description logics restrict the predicate types that can be used

- Unary predicates denote concept membership

Person(x)

- Binary predicates denote roles between instances

hasChild(x, y)

Note on terminology: the DL literature uses slightly different terms to those in RDFS

- Class and concept are interchangeable terms
- Role, relation and property are interchangeable terms

Defining ontologies with Description Logics

Describe classes (concepts) in terms of their necessary and sufficient conditions

Consider an attribute A of a class C :

- Attribute A is a necessary condition for membership of C
 - If an object is an instance of C , then it has A
- Attribute A is a sufficient condition for membership of C
 - If an object has A , then it is an instance of C

Description Logic Reasoning Tasks

Satisfaction

- "Can this class have any instances?"

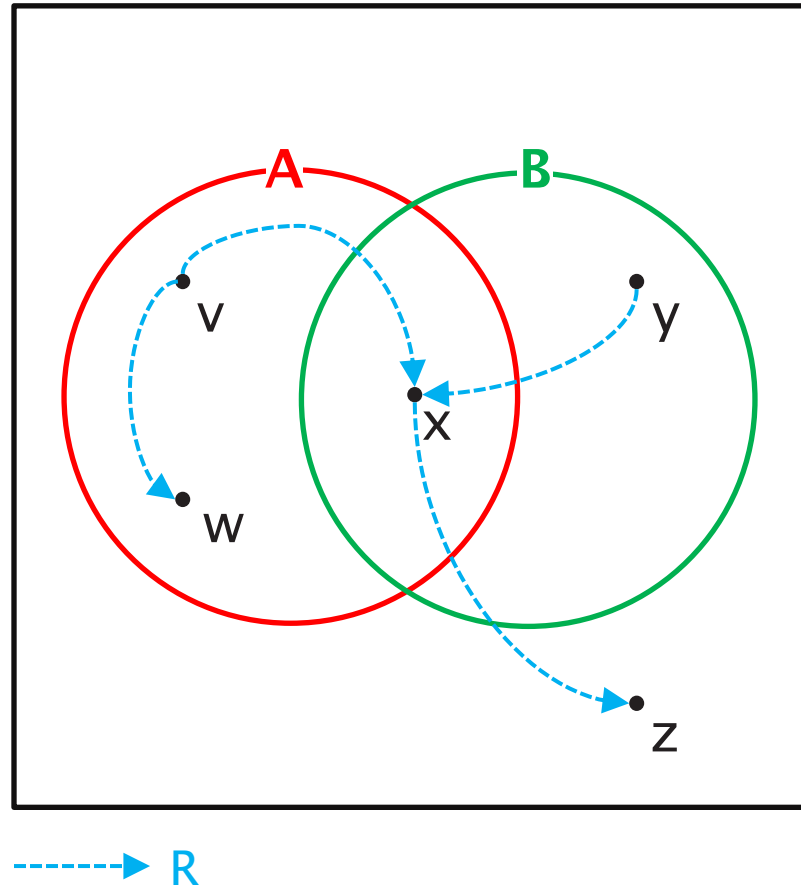
Subsumption

- "Is every instance of class C necessarily an instance of class D?"

Classification

- "What classes is this object an instance of?"

Concepts as sets



Syntax

Expressions

Description logic expressions consist of:

- Concept and role descriptions:
 - Atomic concepts: Person
 - Atomic roles: hasChild
 - Complex concepts: “person with two living parents”
 - Complex roles: “has parent’s brother” (i.e. “has uncle”)
- Axioms that make statements about how concepts or roles are related to each other:
 - “Every person with two living parents is thankful”
 - “hasUncle is equivalent to has parent’s brother”

Concept Constructors

Used to construct complex concepts:

- Boolean concept constructors $\neg C$ $C \sqcup D$ $C \sqcap D$
- Restrictions on role successors $\forall R. C$ $\exists R. C$
- Number/cardinality restrictions $\leq n R$ $\geq n R$ $= n R$
- Nominals (singleton concepts) $\{x\}$
- Universal concept, top \top
- Contradiction, bottom \perp

Role Constructors

Used to construct complex roles:

- Concrete domains (datatypes)
- Inverse roles
- Role composition
- Transitive roles

R^-

$R \circ S$

R^+

OWL and Description Logics

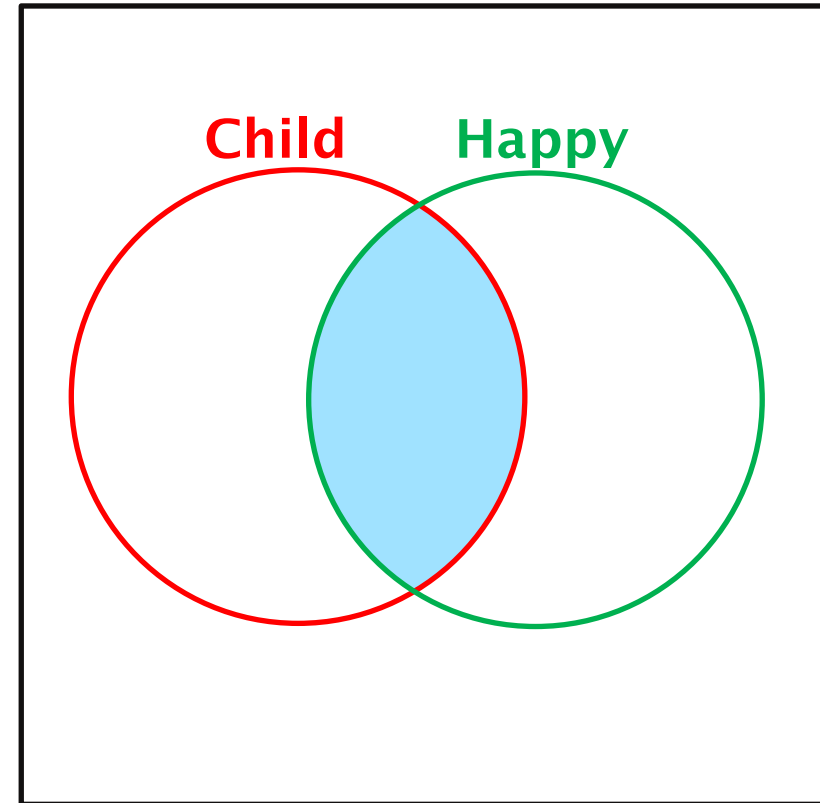
- Not every description logic supports all constructors
- More constructors = more expressive = higher complexity
- For example, OWL DL is equivalent to the logic $\mathcal{SHOIN}(D)$
 - Atomic concepts and roles
 - Boolean operators
 - Universal, existential restrictions, number restrictions
 - Role hierarchies
 - Nominals
 - Inverse and transitive roles (but not role composition)

Boolean Concept Constructors: Intersection

Child \cap Happy

The class of things which are both children and happy

Read as “Child AND Happy”

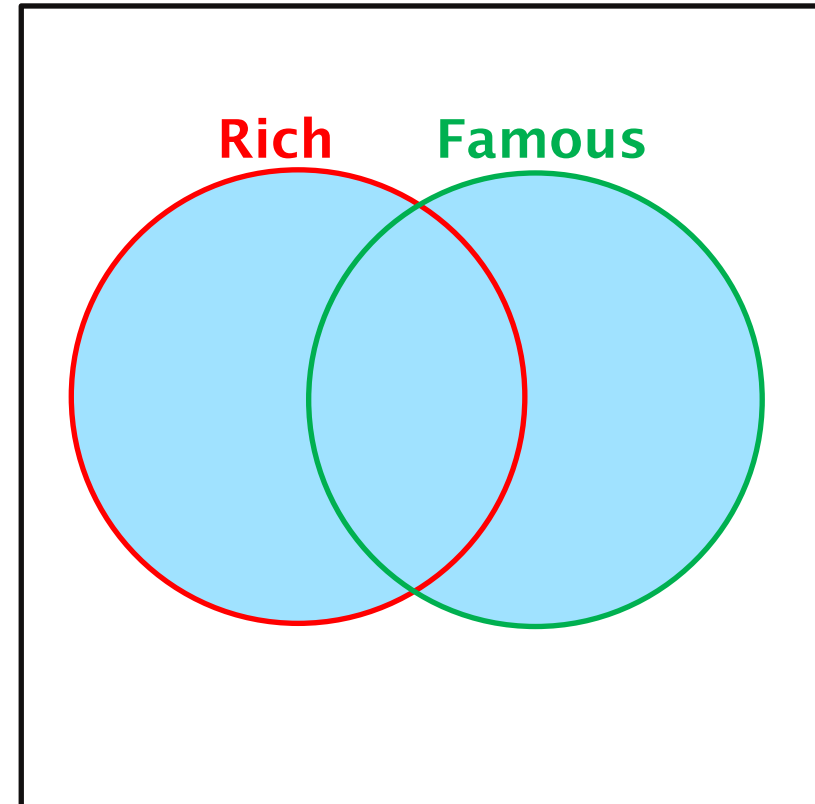


Boolean Concept Constructors: Union

Rich \sqcup Famous

The class of things which are rich or famous
(or both)

Read as “Rich OR Famous”

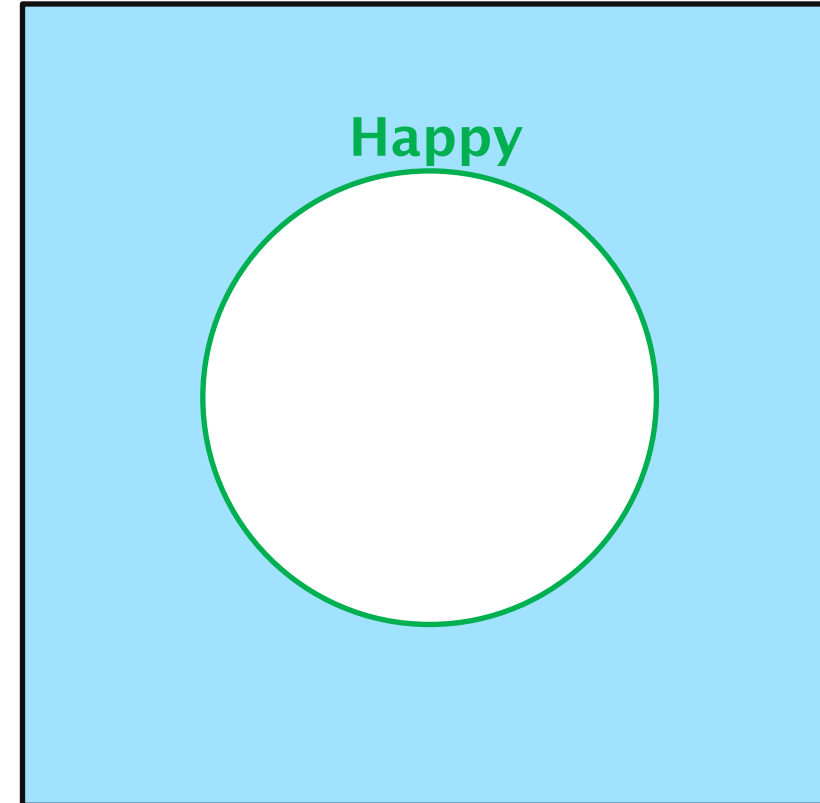


Boolean Concept Constructors: Complement

\neg Happy

The class of things which are not happy

Read as “NOT Happy”



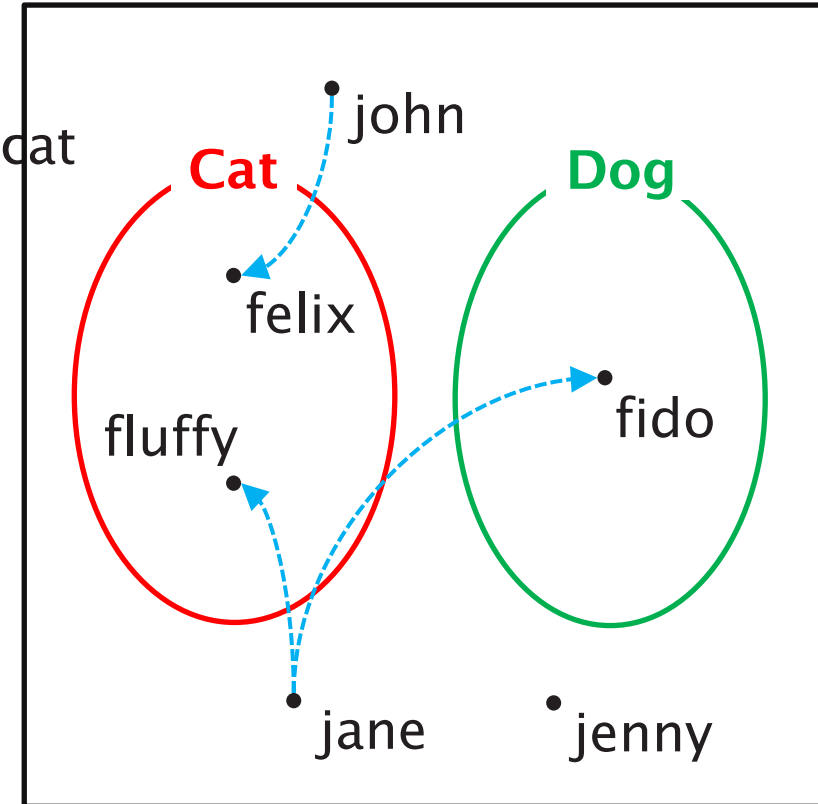
Restrictions: Existential

\exists hasPet. Cat

The class of things which have some pet that is a cat

- must have at least one pet

Read as “hasPet SOME Cat”



-----> hasPet

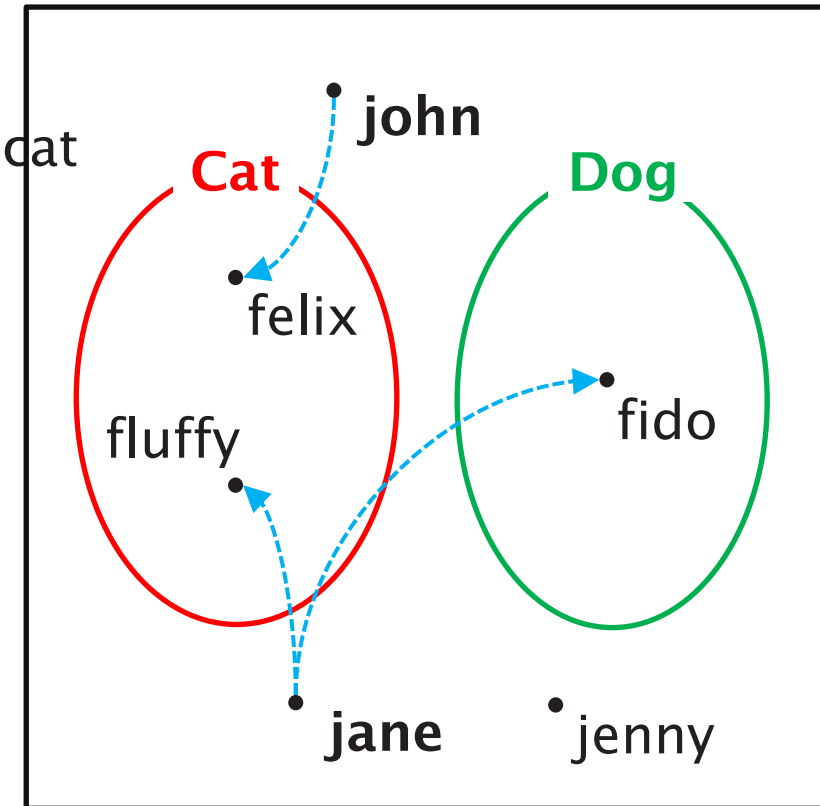
Restrictions: Existential

\exists hasPet. Cat

The class of things which have some pet that is a cat

- must have at least one pet

Read as “hasPet SOME Cat”



-----> hasPet

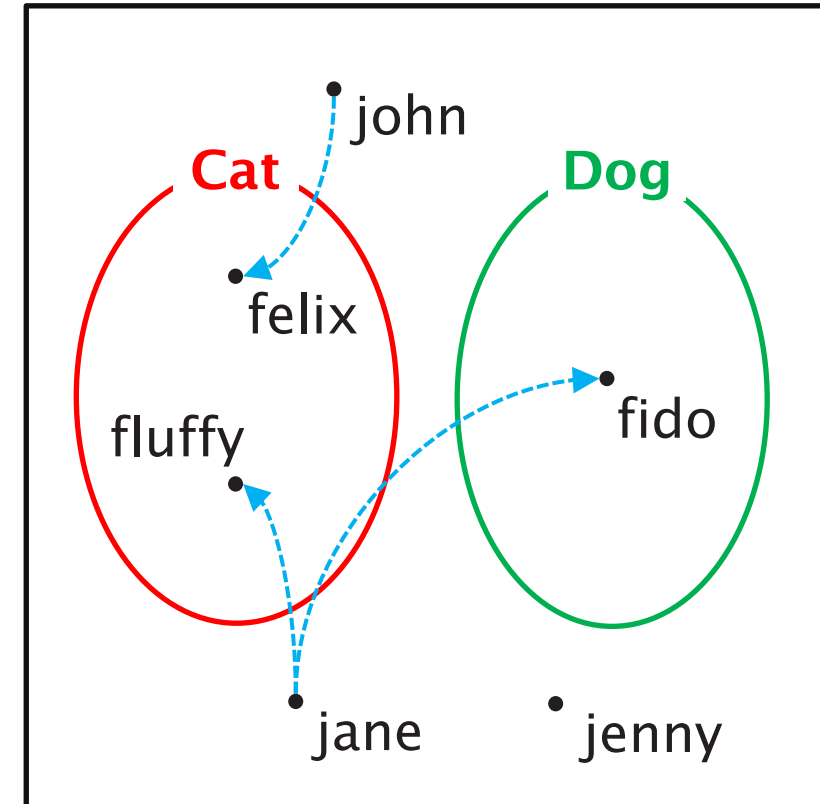
Restrictions: Universal

\forall hasPet. Cat

The class of things all of whose pets are cats

- Or, which only have pets that are cats
- includes those things which have no pets

Read as “hasPet ONLY Cat”



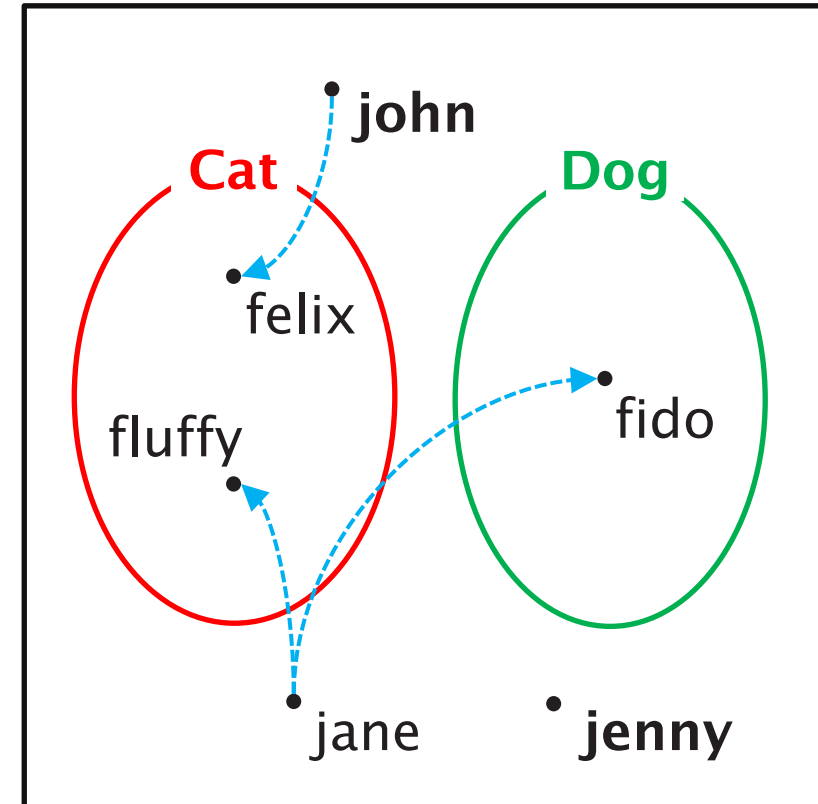
Restrictions: Universal

\forall hasPet. Cat

The class of things all of whose pets are cats

- Or, which only have pets that are cats
- includes those things which have no pets

Read as “hasPet ONLY Cat”



-----> hasPet

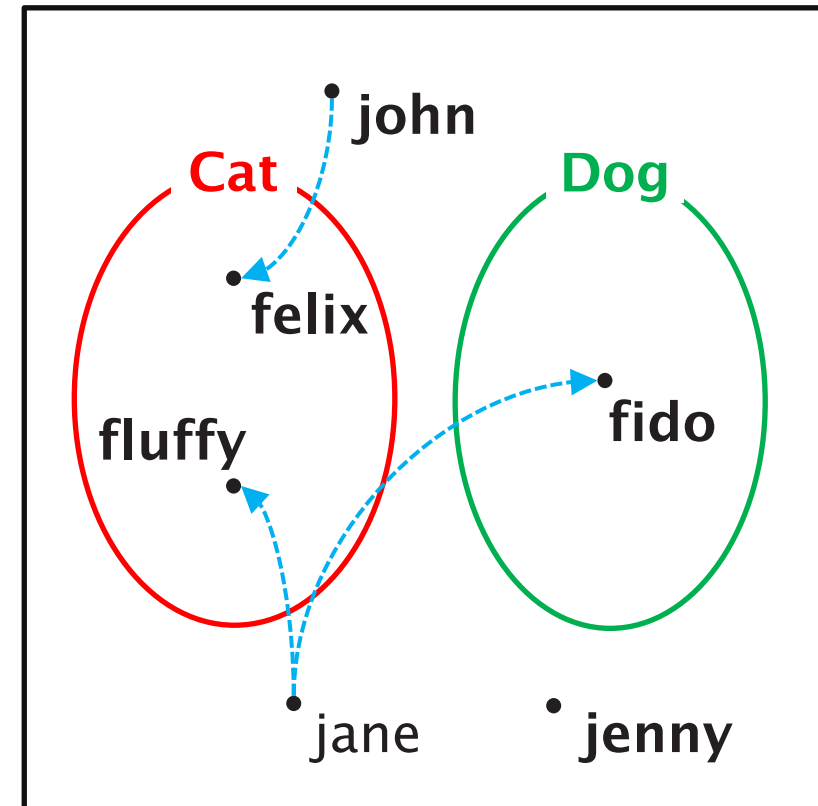
Restrictions: Universal

\forall hasPet. Cat

The class of things all of whose pets are cats

- Or, which only have pets that are cats
- includes those things which have no pets

Read as “hasPet ONLY Cat”

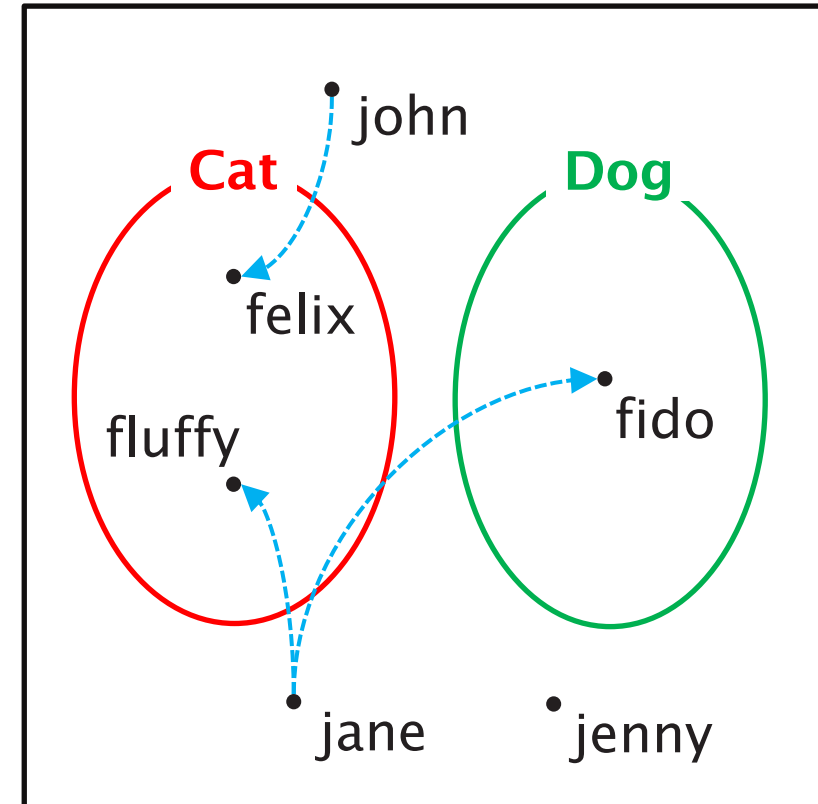


-----> hasPet

Restrictions: Number

= 1 hasPet

The class of things which have exactly one pet

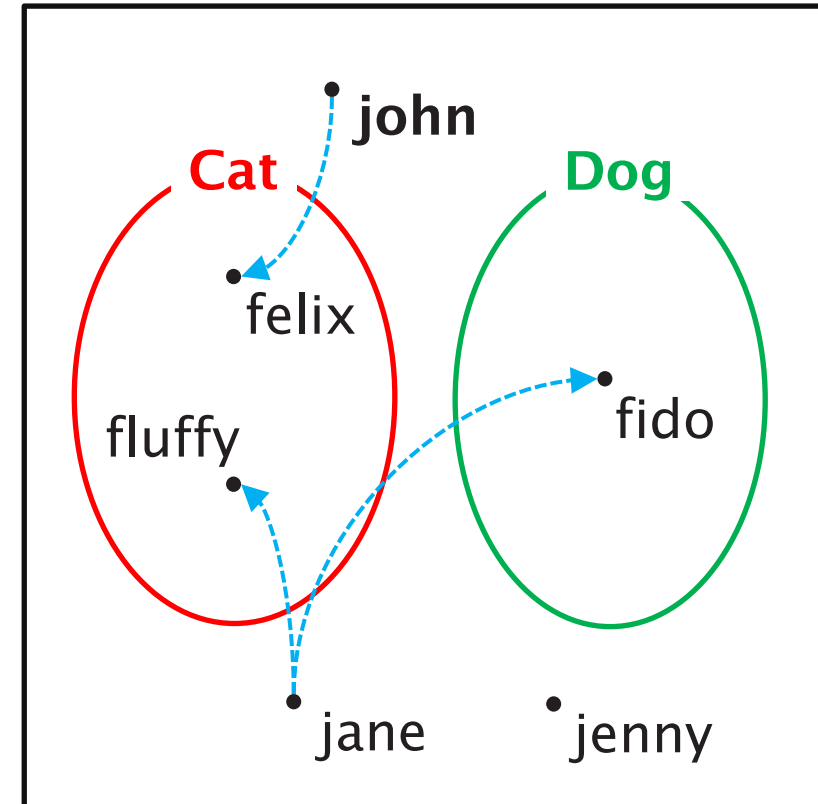


-----> hasPet

Restrictions: Number

= 1 hasPet

The class of things which have exactly one pet

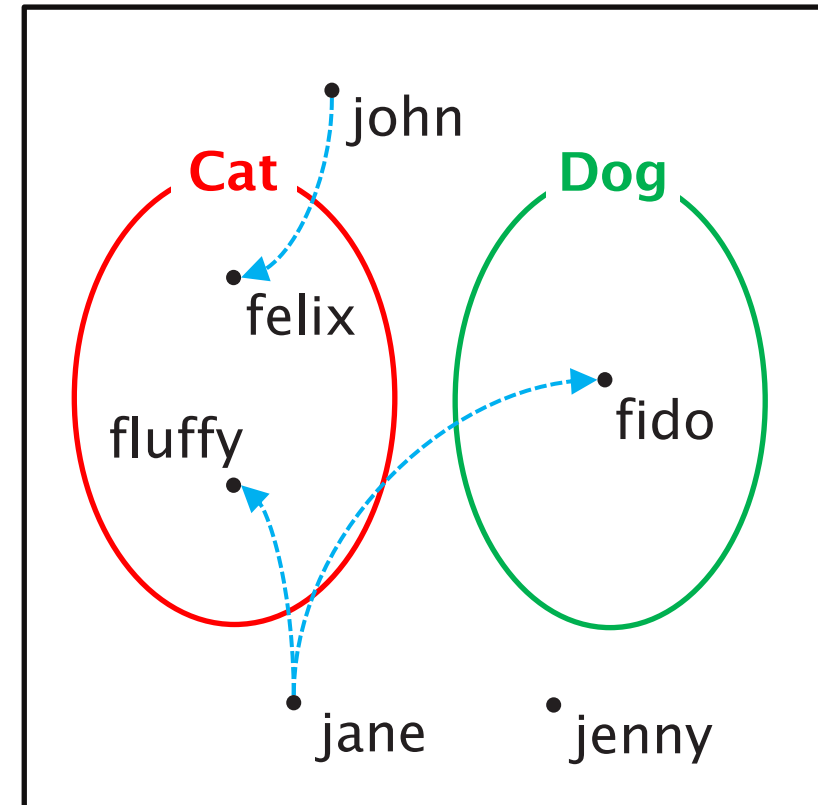


-----> hasPet

Restrictions: Number

≥ 2 hasPet

The class of things which have at least two pets

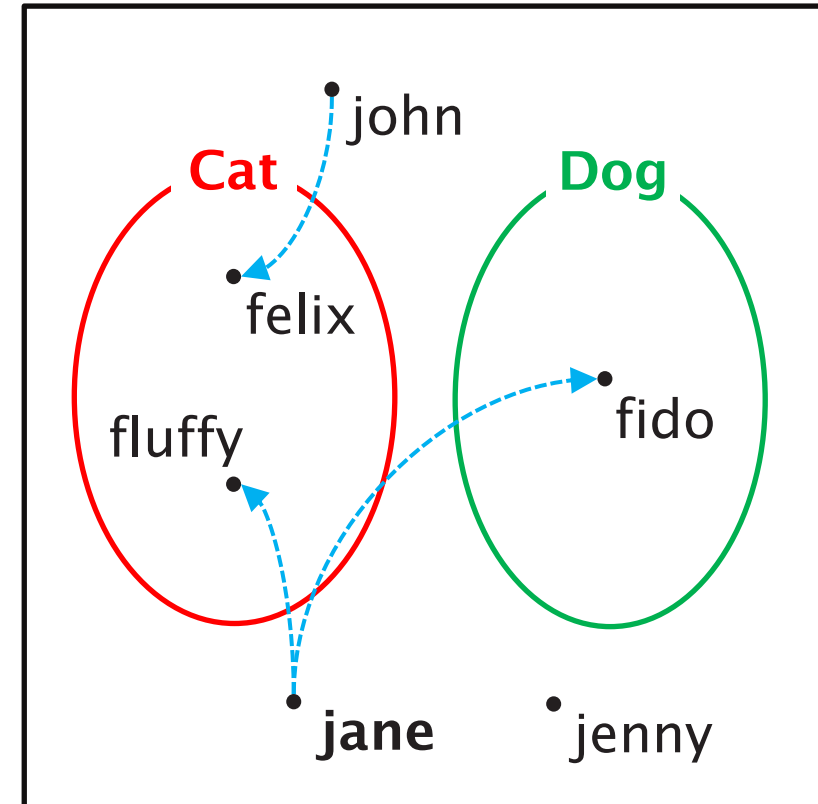


-----> hasPet

Restrictions: Number

≥ 2 hasPet

The class of things which have at least two pets



-----> hasPet

Knowledge Bases

A description logic knowledge base (KB) has two parts:

- TBox: terminology
 - A set of axioms describing the structure of the domain (i.e., a conceptual schema)
 - Concepts, roles
- ABox: assertions
 - A set of axioms describing a concrete situation (data)
 - Instances

TBox Axioms

Concept inclusion
(C is a subclass of D)

$$C \sqsubseteq D$$

Concept equivalence
(C is equivalent to D)

$$C \equiv D$$

Role inclusion
(R is a subproperty of S)

$$R \sqsubseteq S$$

Role equivalence
(R is equivalent to S)

$$R \equiv S$$

Role transitivity
(R composed with itself is a
subproperty of R)

$$R^+ \sqsubseteq R$$

Revisiting Necessary and Sufficient Conditions

“Attribute A is a necessary/sufficient condition for membership of C”

Instead of talking directly about A, we can make a class expression (using the concept constructors) that represents the class of things with attribute A – call it D

- Membership of D is necessary/sufficient for membership of C

Revisiting Necessary and Sufficient Conditions

Membership of D is a necessary condition for membership of C

$$C \subseteq D$$

Membership of D is a sufficient condition for membership of C

$$C \supseteq D$$

Membership of D is both a necessary and a sufficient condition for membership of C

$$C \equiv D$$

Revisiting Necessary and Sufficient Conditions

Some common terminology:

$$C \sqsubseteq D$$

- C is a *primitive* or *partial class*

$$C \equiv D$$

- C is a *defined class*

(you'll see these terms used in the Protégé OWL Tutorial)

ABox Axioms

Concept instantiation

$C(x)$

- x is of type C

Role instantiation

$R(x, y)$

- x has R of y

Axiom Examples

Every person is either living or dead

Every happy child has a loving parent

Every child who eats only cake is
unhealthy

No elephants can fly

A mole is a sauce from Mexico that
contains chili

All Englishmen are mad

Axiom Examples

Every person is either living or dead

$\text{Person} \sqsubseteq \text{Living} \sqcup \text{Dead}$

Every happy child has a loving parent

$\text{Child} \sqcap \text{Happy} \sqsubseteq \exists \text{hasParent. Loving}$

Every child who eats only cake is unhealthy

$\text{Child} \sqcap \forall \text{eats. Cake} \sqcap \exists \text{eats. Cake} \sqsubseteq \neg \text{Healthy}$

No elephants can fly

$\text{Elephant} \sqcap \text{FlyingThing} \equiv \perp$

A mole is a sauce from Mexico that contains chili

$\text{Mole} \equiv$
 $\text{Sauce} \sqcap \exists \text{hasOrigin. \{Mexico\}} \sqcap$
 $\exists \text{hasIngredient. Chili}$

All Englishmen are mad

$\exists \text{bornIn. \{England\}} \sqcap \text{Male} \sqsubseteq \text{Mad}$

Tips for Description Logic Axioms

- No single ‘correct’ answer - different modelling choices
- Break sentence down into pieces
 - e.g. “successful man”, “spicy ingredient” etc
 - Look for nouns and adjectives (concepts)
 - Look for verb phrases (roles)
- Look for indicators of axiom type:
 - “Every X is Y” - inclusion axiom
 - “X is Y” - equivalence axiom
- Remember that $\forall R.C$ is satisfied by instances which have no value for R

**DON'T
PANIC!**

Semantics

Description Logics and Predicate Logic

Description Logics are a subset of first order Predicate Logic with a simplified syntax
Every DL expression can be converted into an equivalent FOPL expression

Description Logics and Predicate logic

Every concept C is translated to a formula $\phi_C(x)$

Every role R is translated to a formula $\phi_R(x, y)$

Boolean concept constructors:

$$\phi_{\neg C}(x) = \neg\phi_C(x)$$

$$\phi_{C \sqcup D}(x) = \phi_C(x) \vee \phi_D(x)$$

$$\phi_{C \sqcap D}(x) = \phi_C(x) \wedge \phi_D(x)$$

Restrictions:

$$\phi_{\exists R.C}(x) = \exists y. \phi_R(x, y) \wedge \phi_C(y)$$

$$\phi_{\forall R.C}(x) = \forall y. \phi_R(x, y) \Rightarrow \phi_C(y)$$

Description Logics and Predicate logic

Axioms are translated as follows:

Concept inclusion $C \sqsubseteq D$

$$\forall x. \phi_C(x) \Rightarrow \phi_D(x)$$

Concept equivalence $C \equiv D$

$$\forall x. \phi_C(x) \Leftrightarrow \phi_D(x)$$

Example

“Every child who eats cake is happy”

Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

Example

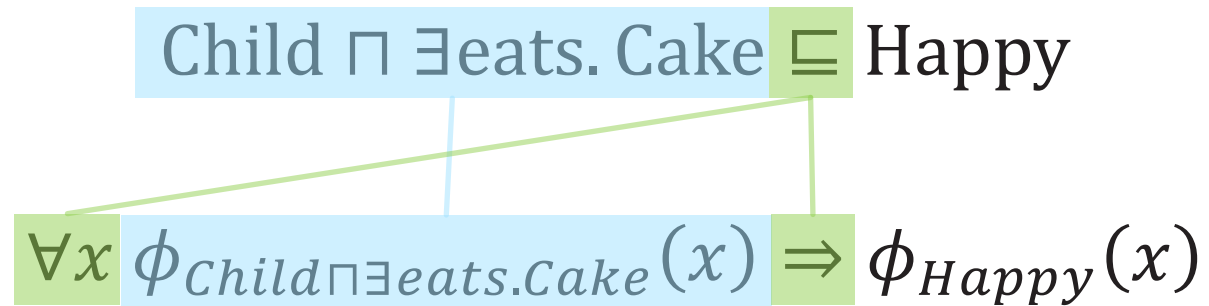
“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

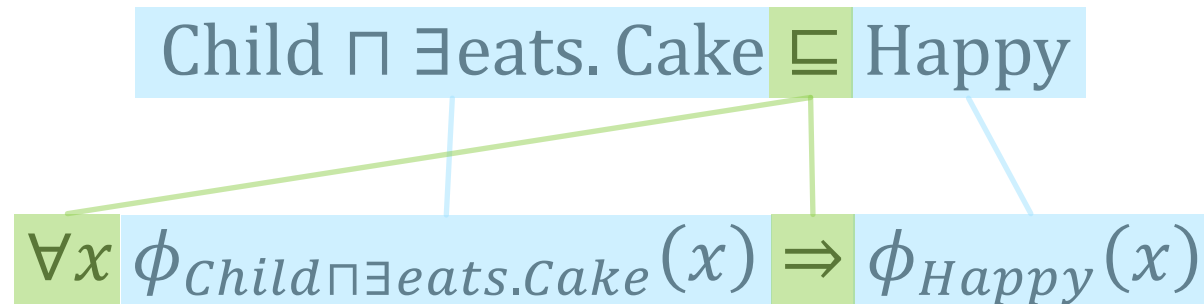
Example

“Every child who eats cake is happy”



Example

“Every child who eats cake is happy”



Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

$$\forall x \phi_{Child}(x) \wedge \phi_{\exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

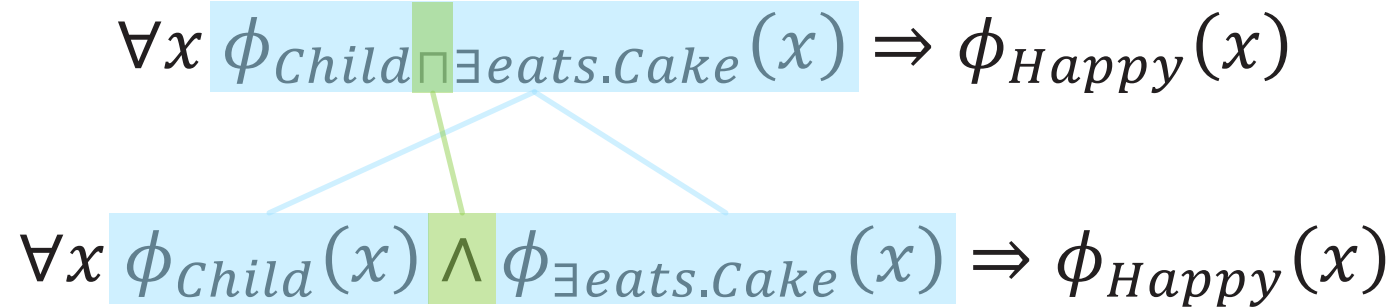
$$\forall x \phi_{Child}(x) \wedge \phi_{\exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

$$\forall x \phi_{Child}(x) \wedge \phi_{\exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$


Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

$$\forall x \phi_{Child}(x) \wedge \phi_{\exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

$$\forall x \phi_{Child}(x) \wedge \exists y \phi_{\text{eats}}(x, y) \wedge \phi_{\text{Cake}}(y) \Rightarrow \phi_{Happy}(x)$$

Example

“Every child who eats cake is happy”

Child \sqcap \exists eats. Cake \sqsubseteq Happy

$$\forall x \phi_{Child \sqcap \exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

$$\forall x \phi_{Child}(x) \wedge \phi_{\exists \text{ eats. Cake}}(x) \Rightarrow \phi_{Happy}(x)$$

$$\forall x \phi_{Child}(x) \wedge \exists y \phi_{\text{eats}}(x, y) \wedge \phi_{\text{Cake}}(y) \Rightarrow \phi_{Happy}(x)$$

Description Logic Semantics

Δ is the domain (non-empty set of individuals)

Interpretation function $\cdot^{\mathcal{J}}$ (or $ext()$) maps:

- Concept expressions to their extensions
(set of instances of that concept, subsets of Δ)
- Roles to subsets of $\Delta \times \Delta$
- Individuals to elements of Δ

Examples:

- $C^{\mathcal{J}}$ is the set of members of C
- $(C \sqcup D)^{\mathcal{J}}$ is the set of members of either C or D

Description Logic Semantics

Syntax	Semantics	Notes
$(C \sqcap D)^J$	$C^J \cap D^J$	Conjunction
$(C \sqcup D)^J$	$C^J \cup D^J$	Disjunction
$(\neg C)^J$	$\Delta \setminus C^J$	Complement
$(\exists R. C)^J$	$\{x \mid \exists y. \langle x, y \rangle \in R^J \wedge y \in C^J\}$	Existential
$(\forall R. C)^J$	$\{x \mid \forall y \langle x, y \rangle \in R^J \Rightarrow y \in C^J\}$	Universal
$(\geq n R)^J$	$\{x \mid \#\{y \mid \langle x, y \rangle \in R^J\} \geq n\}$	Min cardinality
$(\leq n R)^J$	$\{x \mid \#\{y \mid \langle x, y \rangle \in R^J\} \leq n\}$	Max cardinality
$(= n R)^J$	$\{x \mid \#\{y \mid \langle x, y \rangle \in R^J\} = n\}$	Exact cardinality
$(\perp)^J$	\emptyset	Bottom
$(\top)^J$	Δ	Top

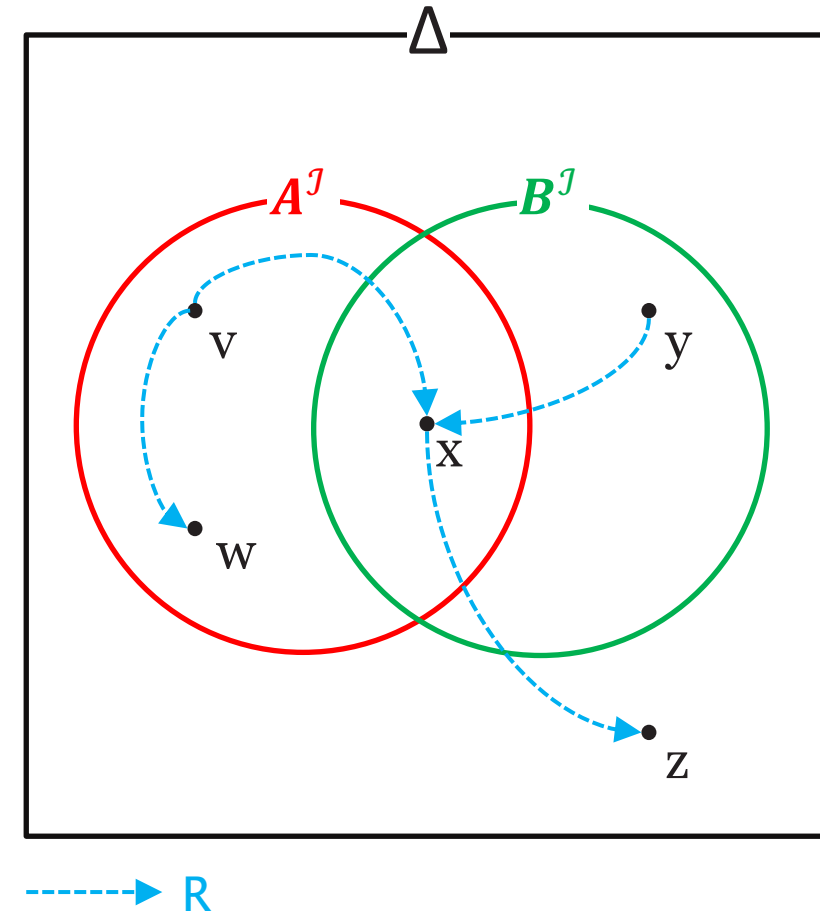
Interpretation Example

$$\Delta = \{v, w, x, y, z\}$$

$$A^J = \{v, w, x\}$$

$$B^J = \{x, y\}$$

$$R^J = \{\langle v, w \rangle, \langle v, x \rangle, \langle y, x \rangle, \langle x, z \rangle\}$$



Interpretation Example

$$(\neg B)^J =$$

$$(A \sqcup B)^J =$$

$$(\neg A \sqcap B)^J =$$

$$(\exists R. B)^J =$$

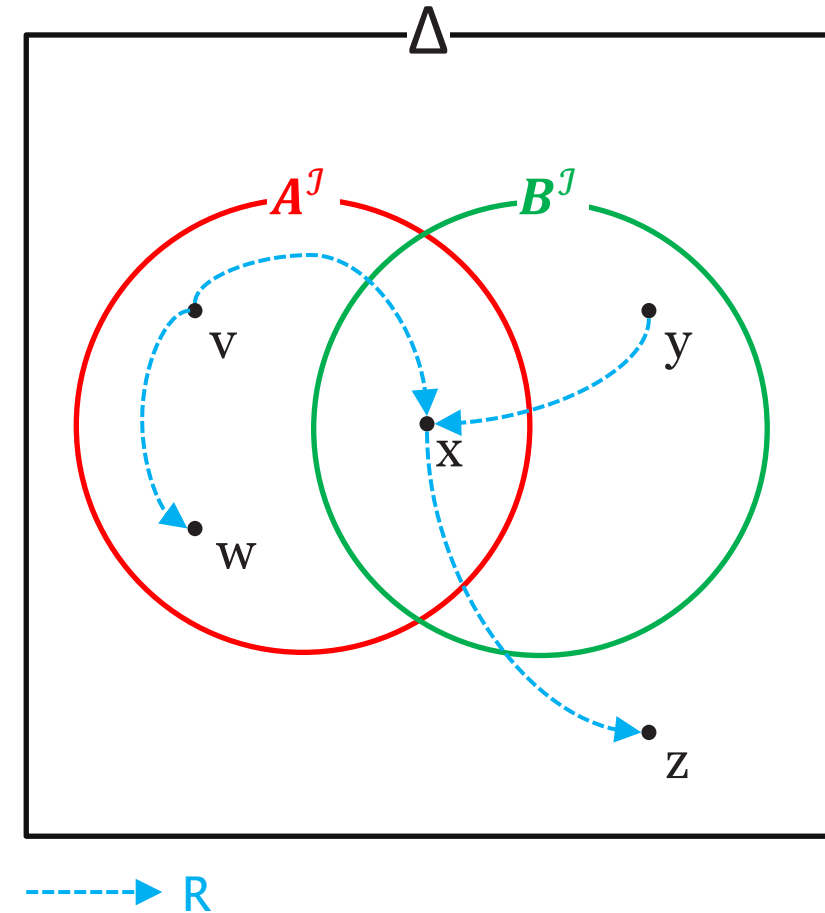
$$(\forall R. B)^J =$$

$$(\exists R. (\exists R. A))^J =$$

$$(\exists R. \neg(A \sqcap B))^J =$$

$$(\exists R^-. A)^J =$$

$$(R^+)^J =$$



Answers

$$(\neg B)^J = \{v, w, z\}$$

$$(A \sqcup B)^J = \{v, w, x, y\}$$

$$(\neg A \sqcap B)^J = \{y\}$$

$$(\exists R. B)^J = \{v, y\}$$

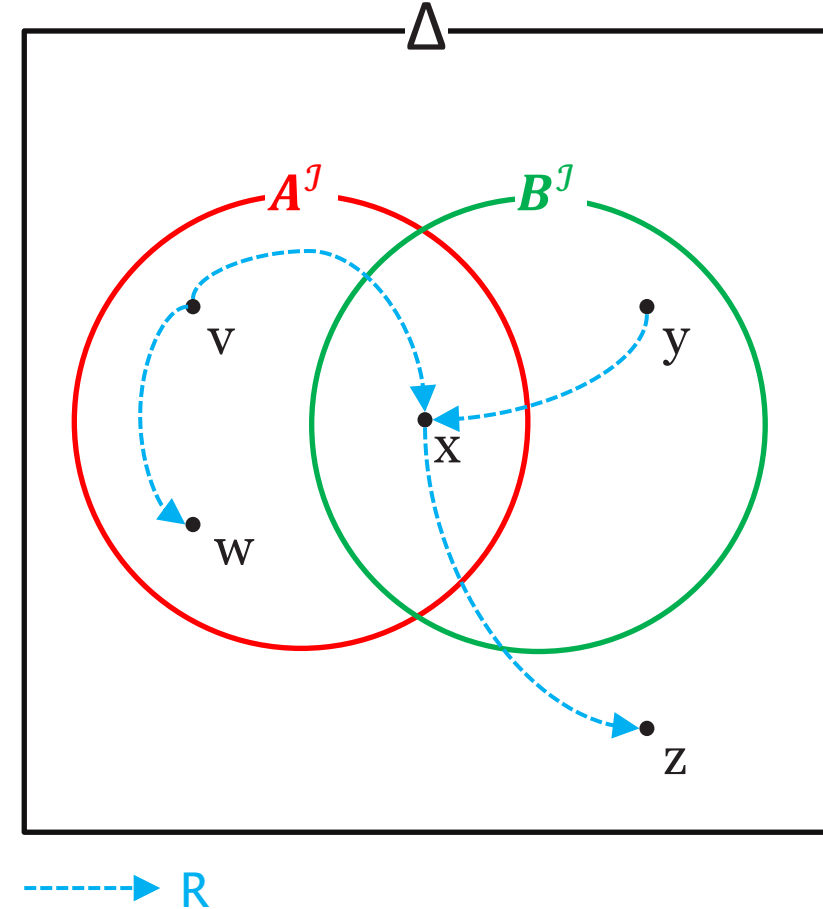
$$(\forall R. B)^J = \{y, w, z\}$$

$$(\exists R. (\exists R. A))^J = \{\}$$

$$(\exists R. \neg(A \sqcap B))^J = \{v, x\}$$

$$(\exists R^- . A)^J = \{w, x, z\}$$

$$(R^+)^J = \{\langle v, w \rangle, \langle v, x \rangle, \langle v, z \rangle, \langle y, x \rangle, \langle y, z \rangle, \langle x, z \rangle\}$$



DL Reasoning Revisited

DL Reasoning Revisited

A description logic knowledge base comprises:

- A TBox defining concepts and roles
- An ABox containing assertions about instances

$$K = \langle TBox, ABox \rangle$$

We can construct an interpretation $\mathcal{I} = \langle \Delta, \cdot^{\mathcal{I}} \rangle$ which maps the instances, concepts and roles in K onto a domain Δ via an interpretation function $\cdot^{\mathcal{I}}$

We can redefine the reasoning tasks in terms of \mathcal{I}

Satisfaction

“Can this class have any instances?”

A class C is satisfiable with respect to a KB K iff there exists an interpretation \mathcal{I} of K with $C^{\mathcal{I}} \neq \emptyset$

Subsumption

“Is every instance of this class necessarily an instance of this other class?”

A class C is subsumed by a class D with respect to a KB K iff
for every interpretation \mathcal{I} of K , $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$

Equivalence

“Is every instance of this class necessarily an instance of this other class, and vice versa?”

A class C is equivalent to a class D with respect to a KB K iff for every interpretation \mathcal{J} of K , $C^{\mathcal{J}} = D^{\mathcal{J}}$

Classification

“Is this individual necessarily an instance of this class?”

An individual x is an instance of class C wrt a KB K iff
for every interpretation \mathcal{I} of K , $x^{\mathcal{I}} \in C^{\mathcal{I}}$

Reduction to Satisfaction

Tableau-based reasoners for description logics (the predominant modern approach) reduce all reasoning tasks to satisfaction:

Subsumption

- C is subsumed by $D \Leftrightarrow (C \sqcap \neg D)$ is unsatisfiable

Equivalence

- C is equivalent to $D \Leftrightarrow$ both $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are unsatisfiable

Classification

- x is an instance of $C \Leftrightarrow (\neg C \sqcap \{x\})$ is unsatisfiable

Further Reading

Daniele Nardi and Ronald J. Brachman (2003) An Introduction to Description Logics, in Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi and Peter F. Patel-Schneider (eds) The Description Logic Handbook: Theory, implementation and applications, Cambridge University Press, 2003, pp.1-40.

F. Baader and W. Nutt (2003) Basic Description Logics, in Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi and Peter F. Patel-Schneider (eds) The Description Logic Handbook: Theory, implementation and applications, Cambridge University Press, 2003, pp.47-100.

Next Lecture: OWL