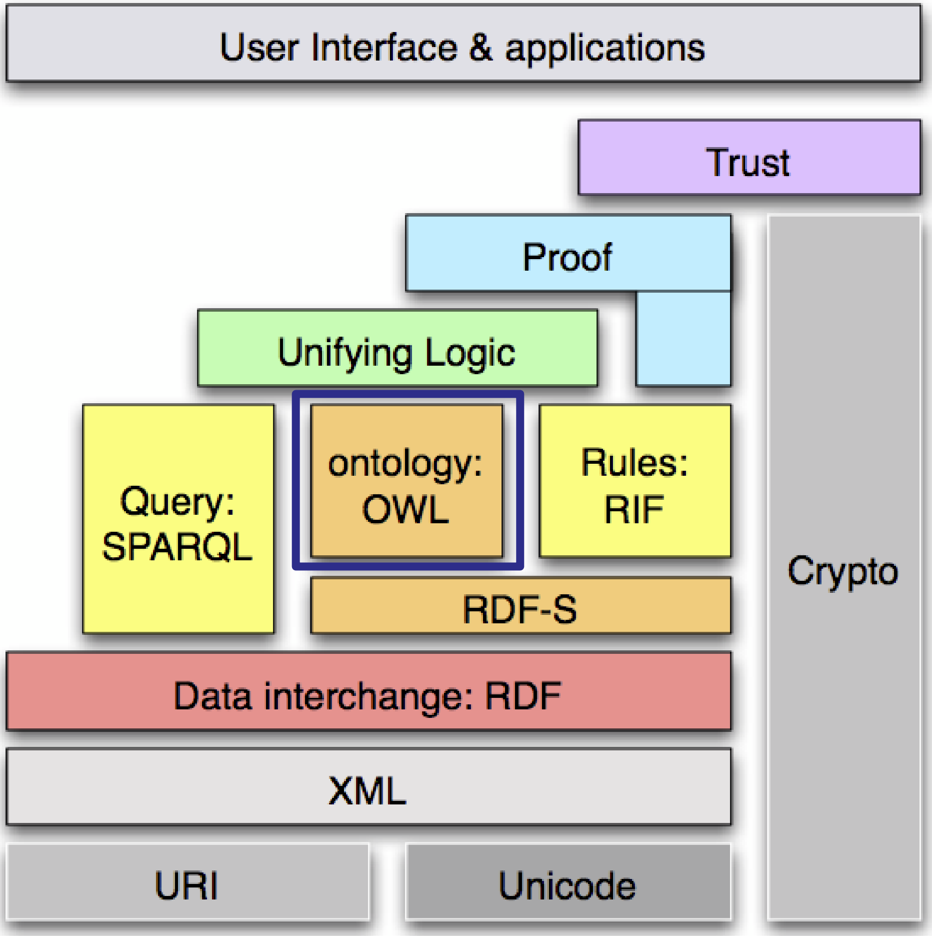
Introduction to Semantic Web Technologies

Kostyantyn Shchekotykhin

Alpen-Adria-Universität

Klagenfurt, Austria

Semantic Web Technology Stack

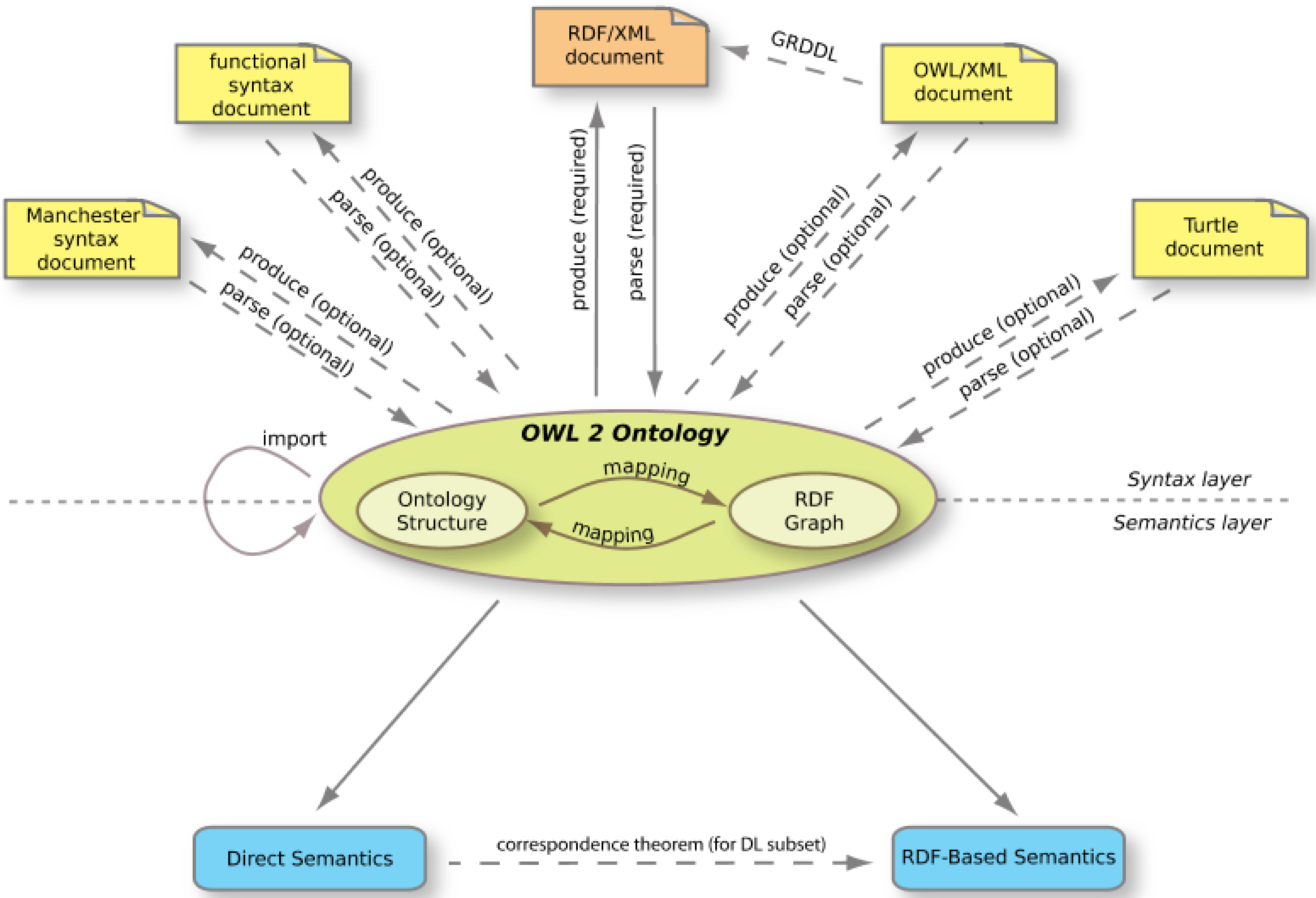


OWL Overview

- OWL – Web Ontology Language
  - W3C Recommendation for the Semantic Web, 2004
  - OWL 2 W3C Recommendation, 2009

- Semantic Web knowledge representation (KR) language based on description logics (DLs)
  - OWL 2 DL (sublanguage of OWL 2) is essentially DL $\mathcal{SROIQ}(\mathcal{D})$
  - KR for Web resources, using URIs.
  - Using web-enabled syntaxes, e.g. based on XML or RDF

The Structure of OWL 2



http://www.w3.org/TR/owl2-overview

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

OWL 2 DL Language Elements

- Fragment of first-order predicate logic (FOL)
- Decidable
- Known complexity classes (N2ExpTime for OWL 2 DL)
- Main language elements
  - Individuals, e.g. *person:peter*

    URIs that correspond to constants in FOL and ressources in RDF
  - Classes (concepts), e.g. *person:Age*

    URIs that correspond to unary predicates in FOL
  - Roles, e.g. person*:hasChild*

    URIs that correspond to binary predicates in FOL and properties in RDF

In the following we will focus only on the OWL 2 DL fragment of OWL 2

Predefined Classes and Properties

- Class *owl:Thing* denoted as ⊤ in DL contains everything
- Class *owl:Nothing* denoted as ⊥ in DL is empty
- Properties *owl:topDataProperty* and *owl:topObjectProperty* connect all possible individuals with all literals/individuals correspondingly
- Properties *owl:bottomDataProperty* and *owl:bottomObjectProperty* do not connect any individuals with a literal/individual correspondingly

DL Syntax

- ABox statements are about individuals (*rdf:type*)

$Person(peter)$ $Person(peter)$

$hasChild(peter, simon)$ $hasChild(peter, simon)$

- TBox statements are about concepts and roles
- Subsumption (*rdfs:subClassOf, rdfs:subPropertyOf)*

$Woman \sqsubseteq Person$ $\forall x(Woman(x) \rightarrow Person(x))$

$hasChild \sqsubseteq hasSuccessor$ $\forall x \forall y(hasChild(x, y) \rightarrow hasSuccessor(x, y)$

- Conjunction (*owl:intersectionOf*)

$Son \sqsubseteq Male \sqcap Child$ $\forall x(Son(x) \rightarrow Male(x) \wedge Child(x))$

- Disjunction (*owl:unionOf*)

$Child \equiv Son \sqcup Daughter$ $\forall x(Child(x) \equiv Son(x) \vee Daughter(x))$

DL Syntax

- Negation (*owl:complementOf*)

$$Daughter \equiv Person \sqcap \neg Son$$

$$\forall x(Daughter(x) \equiv Person(x) \land \neg Son(x))$$

- Existential quantification (*owl:someValuesFrom* property restriction)

$$Son \sqsubseteq \exists hasParent.Person$$

$$\forall x(Son(x) \rightarrow \exists y[hasParent(x, y) \land Person(y)])$$

- Universal quantification (*owl:allValuesFrom* property restriction)

$$HappyParent \equiv Parent \sqcap \forall hasChild.(Person \sqcap Happy)$$

$$\forall x(HappyParent(x) \\ \leftrightarrow Parent(x) \land \forall y[hasChild(x, y) \rightarrow Person(y) \land Happy(y)])$$

$\mathcal{ALC}$ DL

- $\mathcal{AL}$ Attribute Language – the basic DL
- $\mathcal{ALC}$ is the $\mathcal{AL}$ extended with C complex concept negation:
  - ABox assertions
    - individual assignments *Person(simon)*
    - property assignments *hasParent(simon,peter)*
  - TBox assertions that use:
    - subsumption $\sqsubseteq$, equivalence $\equiv$
    - conjunction $\sqcap$, disjunction $\sqcup$, negation $\neg$
    - universal $\forall$ and limited existential $\exists$ property quantification
- $\mathcal{ALC}$ has $EXPTIME$ complexity
  - solvable by a deterministic Turing machine in $O\left(2^{p(n)}\right)$ time, where $p(n)$ is a polynomial depending on the input size
  - $P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$

$\mathcal{ALC}$ Semantics

- The semantics of $\mathcal{ALC}$ is given in terms on interpretations
- An interpretation $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ consists of:
  - non-empty set $\Delta^\mathcal{I}$ called the *domain* of $\mathcal{I}$
  - function $\cdot^\mathcal{I}$ that maps every $\mathcal{ALC}$ concept to a subset of $\Delta^\mathcal{I}$, and every role name to a subset of $\Delta^\mathcal{I} \times \Delta^\mathcal{I}$ such that for all $\mathcal{ALC}$ concepts C, D and all role names r
  - $\top^\mathcal{I} = \Delta^\mathcal{I}$ and $\bot^\mathcal{I} = \emptyset$
  - $(C \sqcap D)^\mathcal{I} = C^\mathcal{I} \cap D^\mathcal{I}$, $(C \sqcup D)^\mathcal{I} = C^\mathcal{I} \cup D^\mathcal{I}$, and $(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \setminus C^\mathcal{I}$
  - $(\exists r.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid Exists\ some\ y \in \Delta^\mathcal{I}\ with\ \langle x, y\rangle \in r^\mathcal{I}\ and\ y \in C^\mathcal{I}\}$
  - $(\forall r.C)^\mathcal{I} = \{x \in \Delta^\mathcal{I} \mid Forall\ y \in \Delta^\mathcal{I}\ if\ \langle x, y\rangle \in r^\mathcal{I}\ then\ y \in C^\mathcal{I}\}$
  - $C^\mathcal{I}$ ($r^\mathcal{I}$) is called the extension of the concept C (role name r) in the interpretation $\mathcal{I}$
  - If some $\mathrm{x} \in C^\mathcal{I}$ then x is an instance of C in $\mathcal{I}$

Reasoning for $\mathcal{ALC}$

- All reasoning tasks for $\mathcal{ALC}$ can be reduced to verification of inconsistency of the knowledge base
- Tableaux algorithms are mostly used modern DL reasoners

- Given: KB – TBox $\mathcal{T}$ and ABox $\mathcal{A}$ in the negation normal form (NNF)
- Output: model of the KB or inconsistent

- The algorithm works on a data structure called a *completion forest*, which is a directed labeled graph.
- Each node of this graph is a root of *completion tree*, where:
  - nodes are individuals or variable names
  - each node x is labeled with a set of classes $\mathcal{L}(x)$
  - each edge $\langle x, y \rangle$ is labeled with a set of role names $\mathcal{L}(\langle x, y \rangle)$

Reasoning for $\mathcal{ALC}$ I

- The forest is initialized such that:
  - it contains a node x_a for each individual $a \in \mathcal{A}$, with $\mathcal{L}(x_a) = \{C | a{:}\, C \in \mathcal{A}\}$
  - for each pair (a, b) of individuals for which $\{r | (a, b){:}\, r \in \mathcal{A}\} \neq \emptyset$ there is an edge $\langle x_a, x_b \rangle$ labeled with $\mathcal{L}(\langle x_a, x_b \rangle) = \{r | (a, b){:}\, r \in \mathcal{A}\}$

- The algorithm applies expansion rules that syntactically decompose the concepts in node labels:
  - infer new constrains for a given node
  - extend the tree according to the constraints

Reasoning for $\mathcal{ALC}$ II

- Blocking prevents applications of expansion rules when the construction becomes repetitive. A node x is blocked if
  - there is an ancestor y of x such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$
  - there is an ancestor z of x such that z is blocked

- The algorithm stops if it encounters a clash, i.e. there is a node x for which $\{A, \neg A\} \subseteq \mathcal{L}(x)$

The tableaux expansion rules for $\mathcal{ALC}$

$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

$\exists$-rule: if 1. $\exists r.C \in \mathcal{L}(x)$, x is not blocked, and
2. x has no r-successor y with $C \in \mathcal{L}(y)$,
then create a new node y with $\mathcal{L}(\langle x, y\rangle) = \{r\}$ and $\mathcal{L}(y) = \{C\}$

$\forall$-rule: if 1. $\forall r.C \in \mathcal{L}(x)$, x is not blocked, and
2. there is an r-successor y of x with $C \notin \mathcal{L}(y)$
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\sqsubseteq$-rule: if 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x is not blocked, and
2. $C_2 \sqcup \dot{\neg} C_1 \notin \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \dot{\neg} C_1\}$

Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In *Handbook of Knowledge Representation*. Elsevier, 2007.

Reasoning example

Given the following KB verify if it entails $Alive(minki)$

$HappyCatOwner(schrödinger)$

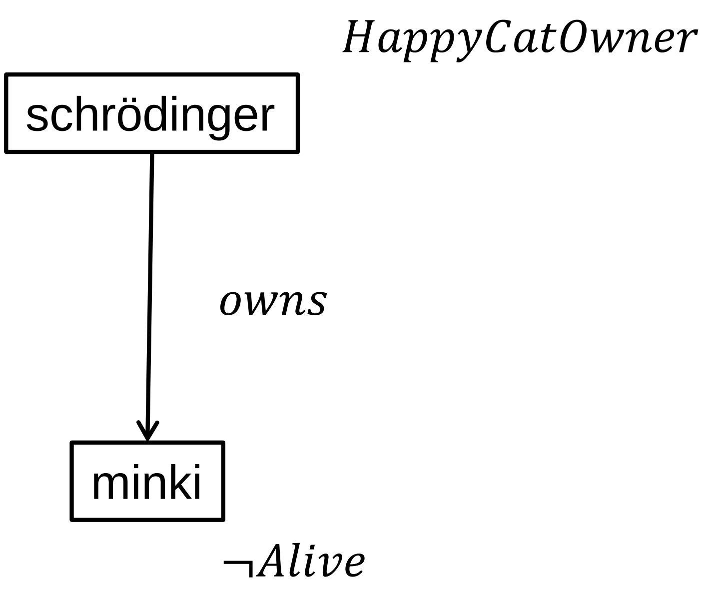
$owns(schrödinger, minki)$

$HappyCatOwner \sqsubseteq \forall owns.(Cat \sqcap \neg Dead)$

$Cat \sqsubseteq Dead \sqcup Alive$

Initialize the forest:

- Create nodes for individuals
- Label them with classes that they belong to
- Create edges corresponding to given roles
- Label edges with role names



Reasoning example

$HappyCatOwner \sqsubseteq \forall owns.(Cat \sqcap \neg Dead)$

$Cat \sqsubseteq Dead \sqcup Alive$

rule 5

schrödinger

$HappyCatOwner$

$\neg HappyCatOwner \sqcup \forall owns.(Cat \sqcap \neg Dead)$

rule 5

$owns$

$\sqsubseteq$-rule: if 1. $C_1 \sqsubseteq C_2 \in \mathcal{T}$, x is not blocked, and
2. $C_2 \sqcup \dot{\neg} C_1 \notin \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2 \sqcup \dot{\neg} C_1\}$

minki

$\neg Alive$

$\neg Cat \sqcup Dead \sqcup Alive$

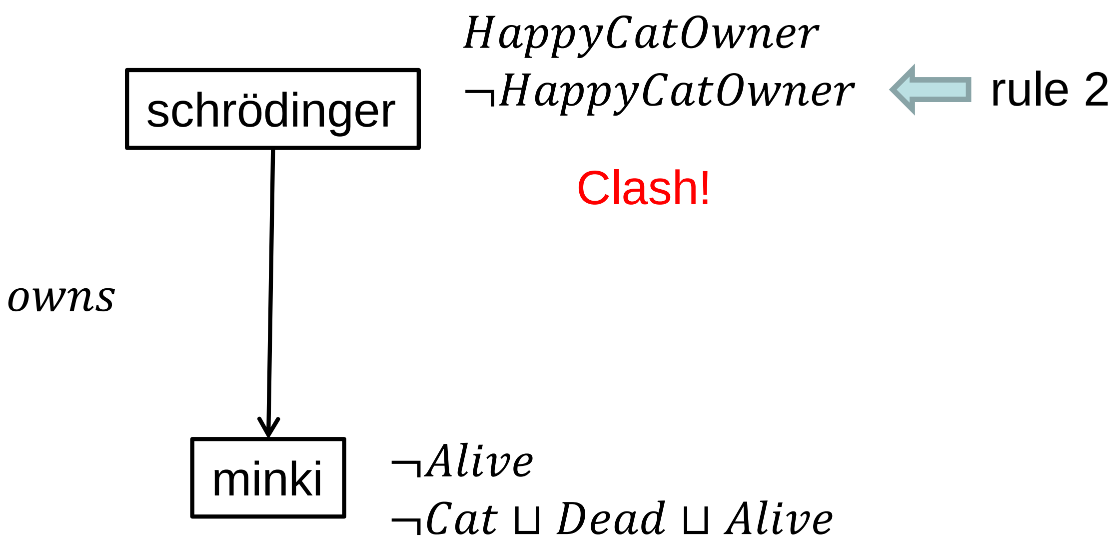
Note that redundant applications of the rule 5 are omitted

Reasoning example

Consider the set $\mathcal{L}(schrödinger)$ containing a fact and a disjunction

$$\neg HappyCatOwner \sqcup \forall owns.(Cat \sqcap \neg Dead)$$

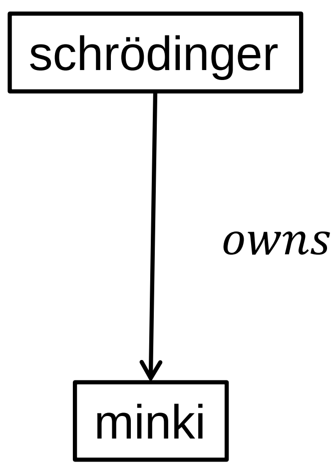
where either $\neg HappyCatOwner$ is true or $\forall owns.(Cat \sqcap \neg Dead)$ is true or both



$\sqcup$-rule: if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C\}$ for some $C \in \{C_1, C_2\}$

Reasoning example

The algorithm backtracks and applies rule 2 to select the other disjunct



$HappyCatOwner$

$\forall owns.(Cat \sqcap \neg Dead)$ ⇐ rule 2

$\neg Alive$

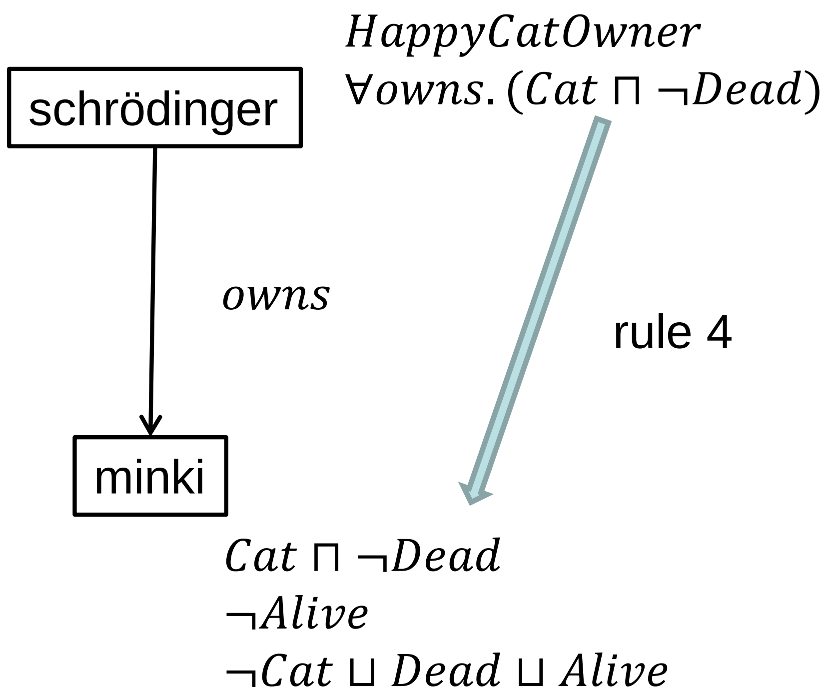
$\neg Cat \sqcup Dead \sqcup Alive$

Reasoning example

Rule 4 $\forall$-rule: if 1. $\forall r.C \in \mathcal{L}(x)$, x is not blocked, and

2. there is an r-successor y of x with $C \notin \mathcal{L}(y)$

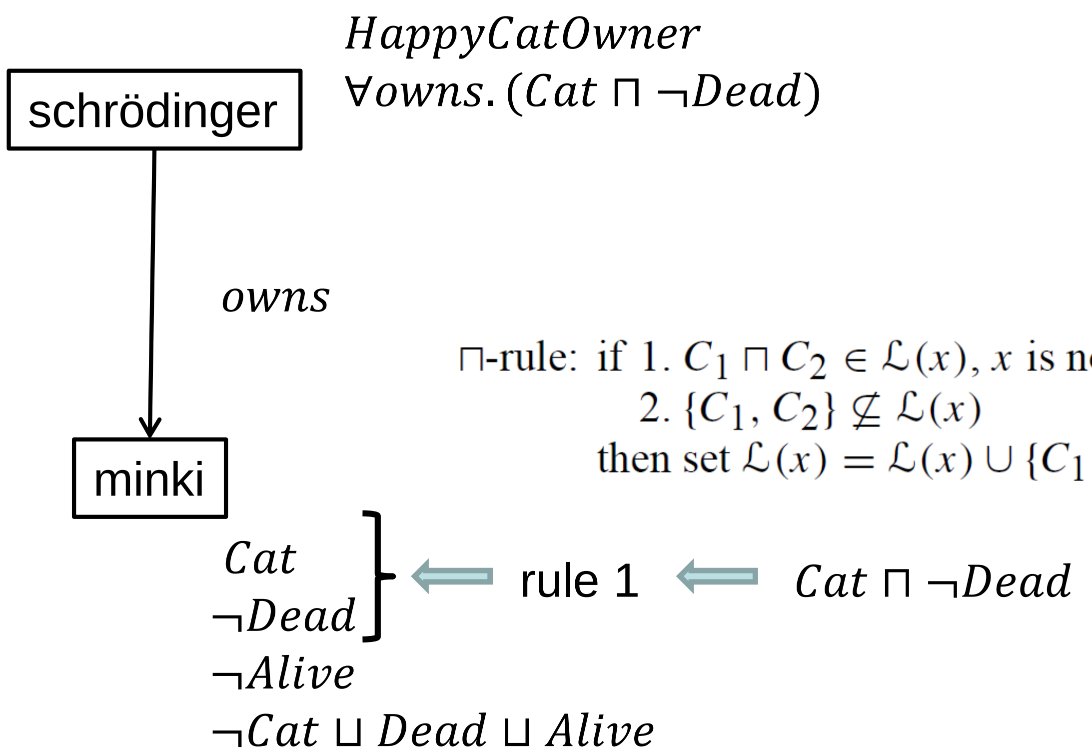
then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$



Reasoning example

Application of the rule 2 to the disjunction allows two possibilities, namely $\neg Dead$ and $Alive$

Both possibilities result in clashes!



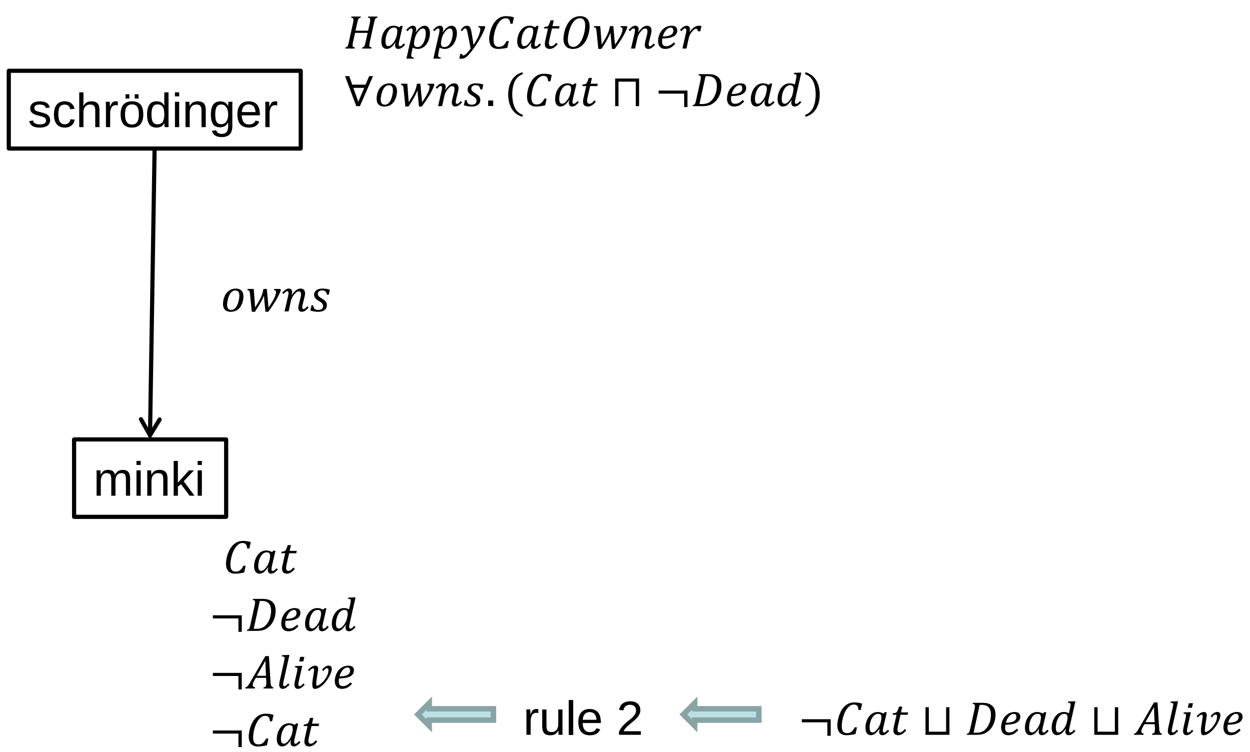
$\sqcap$-rule: if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and
2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$
then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT | WIEN GRAZ

Reasoning example

Application of the rule 2 to the disjunction allows two possibilities, namely $\neg Dead$ and $Alive$

Both possibilities result in clashes!



Clash! In this case and for all other choices of rule 2

ALPEN-ADRIA UNIVERSITÄT KLAGENFURT I WIEN GRAZ

$\mathcal{SROIQ}(\mathcal{D})$ DL

- $\mathcal{ALC}$ + role chains = $\mathcal{SR}$ (*owl:propertyChainAxiom*)

  $hasSister \circ hasSon \sqsubseteq hasNephew$

  $\forall x \forall y \left(\exists z \left(hasSister(x,z) \circ hasSon(z,y)\right) \rightarrow hasNephew(x,y)\right)$

  - includes both top and bottom properties
- includes $\mathcal{S} - \mathcal{ALC}$ with transitive roles

  $isOlder \circ isOlder \sqsubseteq isOlder$

- includes $\mathcal{SH} - \mathcal{S}$ with role hierarchies (*rdf:subPropertyOf*)

  $hasSon \sqsubseteq hasChild$

- $\mathcal{R}$ stands for limited complex role inclusion axioms, role disjointness

$\mathcal{SROIQ}(\mathcal{D})$ DL

- $\mathcal{O}$ stands for nominals, i.e. closed classes

  $PeterChildren \equiv \{simon, daniela\}$

  this not the same as

  $PeterChildren(simon)$

  $PeterChildren(daniela)$

- Individual equality and inequality

  $Peter = Jonson \qquad \{peter\} \equiv \{jonson\}$

  $Peter \neq Jonson \qquad \{peter\} \sqcap \{jonson\} \equiv \bot$

$\mathcal{SROIQ}(\mathcal{D})$ DL

- $\mathcal{I}$ stands for inverse roles

  $hasPredecessor \sqsubseteq hasSuccessor^{-}$

- $\mathcal{Q}$ stands for qualified cardinality restrictions

  $ManyChildren \equiv \geq 4\ hasChild.Person$ owl:minQualifiedCardinality

  $OneChild \equiv = 1\ hasChild.Person$ owl:QualifiedCardinality

  $PC \sqsubseteq \leq 2\ hasComponent.Processor$ owl:maxQualifiedCardinality

- $(\mathcal{D})$ indicates that DL supports use of datatype properties, data values or data types
- OWL2 supports all XSD 1.1 datatypes

  $Car \sqsubseteq Vehicle \sqcap \exists hasWheels.(xsd{:}integer \geq 4\ and \leq 6)$

  OWL only, not a standard DL definition!

$\mathcal{SROIQ}(\mathcal{D})$ DL

- Property chains restrictions required to prevent undecidability
  - there must be a strict linear order $\prec$ on the properties
  - every chain must be of one of the following forms ($s_i \prec r,\ i = 1 \ldots n$)

1) $r \circ r \sqsubseteq r$, 2) $r \circ s_1 \circ .. \circ s_n \sqsubseteq r$,

3) $s_1 \circ .. \circ s_n \circ r \sqsubseteq r$, 4) $s^- \sqsubseteq r$,

5) $s_1 \circ .. \circ s_n \sqsubseteq r$

- Combining property chains with cardinality and self constraints may lead to undecidability. One have to use only those properties in cardinality expressions, which cannot be directly or indirectly inferred from property chains.
- For example:
  - the property r defined as $s_1 \circ .. \circ s_n \sqsubseteq r,\ n > 1$ cannot be used with cardinality constraints
  - the same true for the property $t \sqsubseteq r$

Additional OWL Constructs

- Disjoint classes (*owl:disjointWith)*
- Disjoint union
- Domains (*rdf:domain*) and ranges (*rdf:range*) of properties
- Self (*owl:hasSelf* “true”^^xsd:boolean)

$$SelfMadeMan \equiv Man \sqcap \exists hasMade.Self$$

- Other property characteristics expressible in OWL: (inverse) functionality, transitivity, symmetry, asymmetry, reflexivity, and irreflexivity

OWL 2 Profiles

- Non-determinism is the main source of intractability: $\sqcup$, or $\neg$ with $\sqcup$, etc.
- OWL 2 sublanguages that have $PTIME$ (polynomial time) complexity
  - *OWL 2 EL* is particularly useful in applications employing ontologies that contain very large numbers of properties and/or classes.
  - *OWL 2 QL* is aimed at applications that use very large volumes of instance data, and where query answering is the most important reasoning task.
  - *OWL 2 RL* is aimed at applications that require scalable reasoning without sacrificing too much expressive power.