Description Logic: A Formal Foundation for Ontology Languages and Tools

Part 2: Tools

Ian Horrocks
<ian.horrocks@comlab.ox.ac.uk>
Information Systems Group
Oxford University Computing Laboratory

Contents

- Motivation for Description Logic reasoning
- Basic reasoning tasks/problems
- Reasoning techniques
  - Tableau
  - Completion
  - Query rewriting
  - Rule-based
- Other reasoning tasks
- Recent and future work

Description Logic Reasoning

What Are Description Logics?

- Modern DLs (after Baader et al) distinguished by:
  - Fully fledged logics with formal semantics
    - Decidable fragments of FOL (often contained in C_2)
    - Closely related to Propositional Modal/Dynamic Logics & Guarded Fragment
  - Computational properties well understood (worst case complexity)
  - Provision of inference services
    - Practical decision procedures (algorithms) for key problems (satisfiability, subsumption, query answering, etc)
    - Implemented systems (highly optimised)

What Are Description Logics?

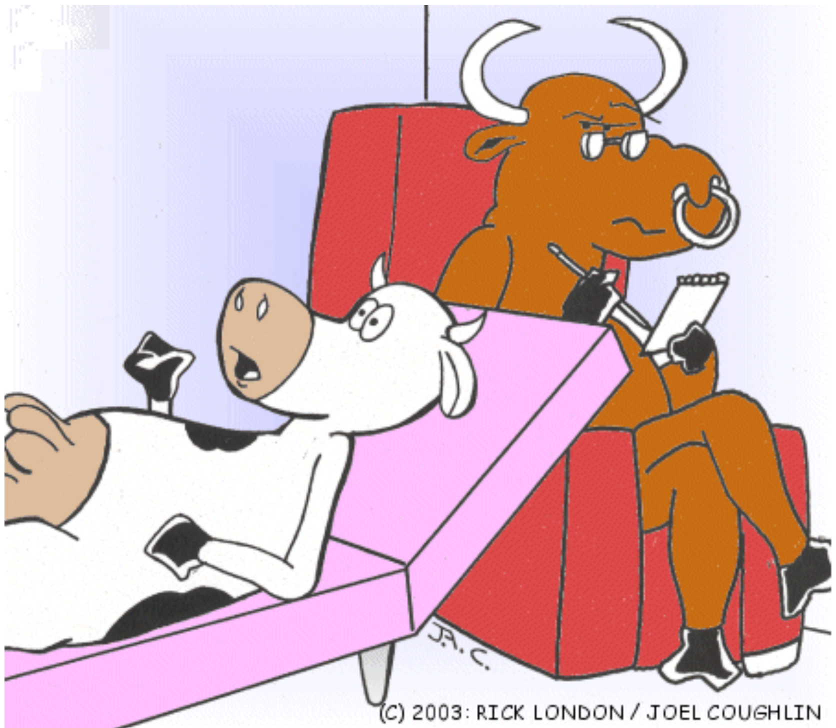
- Modern DLs (after Baader et al) distinguished by:
  - Fully fledged logics with formal semantics
    - Decidable fragments of FOL (often contained in C_2)
    - Closely related to Propositional Modal/Dynamic Logics & Guarded Fragment
  - Computational properties well understood (worst case complexity)
  - **Provision of inference services**
    - Practical decision procedures (algorithms) for key problems (satisfiability, subsumption, query answering, etc)
    - Implemented systems (highly optimised)

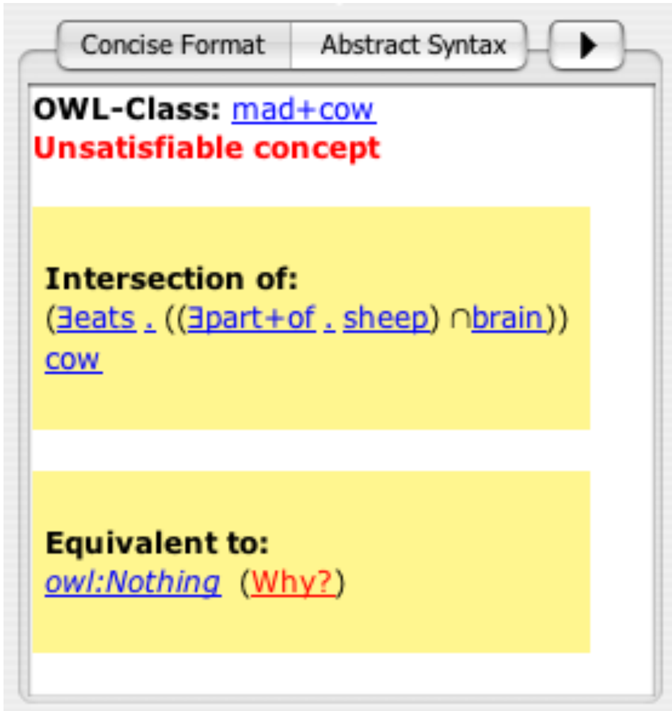
Why Ontology Reasoning?

- Developing and maintaining quality ontologies is ***hard***

Why Ontology Reasoning?

- Developing and maintaining quality ontologies is ***hard***
- Reasoners allow domain experts to check if, e.g.:
  - classes are consistent (no "obvious" errors)



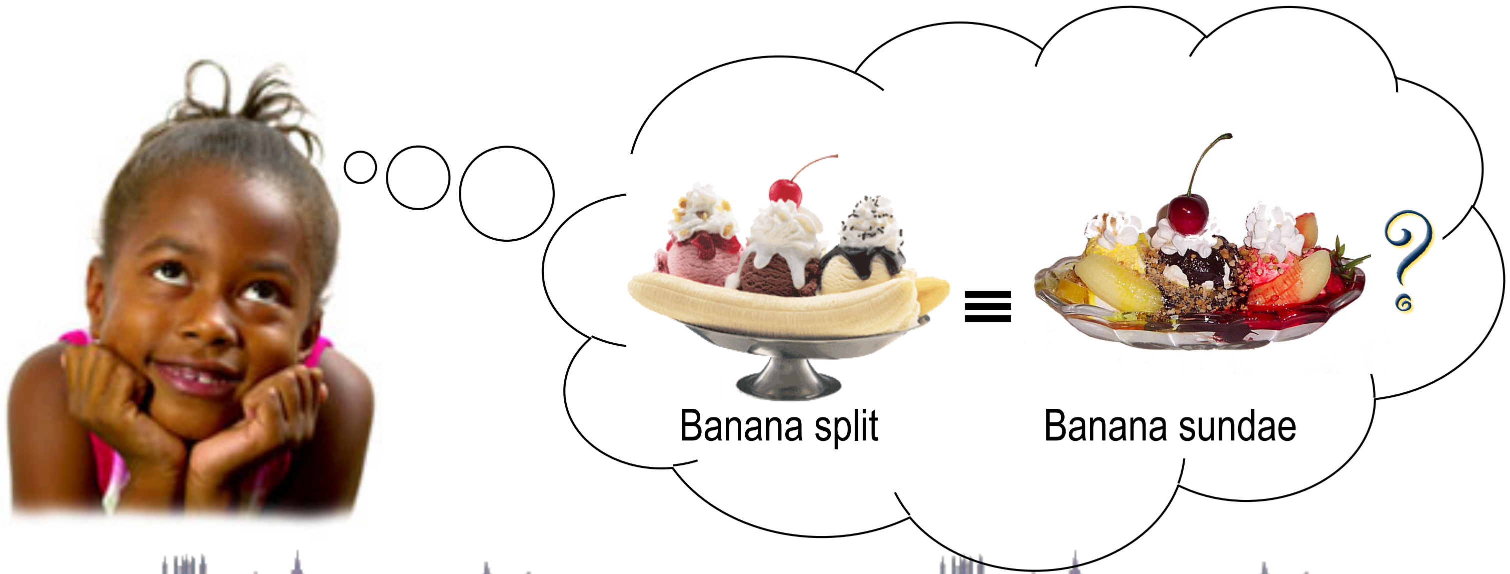


Why Ontology Reasoning?

- Developing and maintaining quality ontologies is ***hard***
- Reasoners allow domain experts to check if, e.g.:
  - classes are consistent (no “obvious” errors)
  - expected subsumptions hold (consistent with intuitions)

Why Ontology Reasoning?

- Developing and maintaining quality ontologies is ***hard***
- Reasoners allow domain experts to check if, e.g.:
  - classes are consistent (no “obvious” errors)
  - expected subsumptions hold (consistent with intuitions)
  - unexpected equivalences hold (unintended synonyms)



Basic Reasoning Tasks

- **Using ontologies** in applications is also very challenging
  - TBox (schema) may be large
  - Abox (data) may be very large
  - Query answers may depend on interactions between schema & data
- **Query answering**
  - Is the parent of a Doctor necessarily a HappyParent? (schema)
  - Is John a HappyParent? (schema + data)
  - Retrieve all instances of Wizards having pet Owls (schema + data)

Basic Reasoning Problem

- Is an axiom/fact **entailed** by ontology/KB
  - Ontology contains obvious errors

    $\mathcal{K} \models C \equiv \bot$ for some concept name C ?
  - Ontology is consistent with intuitions

    $\mathcal{K} \models C \sqsubseteq D$ s.t. expert believes $C \not\sqsubseteq D$?
    $\mathcal{K} \models C \not\sqsubseteq D$ or $\mathcal{K} \models C \sqsubseteq D$ s.t. expert believes $C \sqsubseteq D$?
  - Ontology entails unexpected equivalences

    $\mathcal{K} \models C \equiv D$ for concept names C and D ?
  - Ontology entails query answers

    $\mathcal{K} \models (\text{Parent} \sqcap \exists \text{hasChild.Doctor}) \sqsubseteq \text{HappyParent}$?
    $\mathcal{K} \models \text{John:HappyParent}$?
    Retrieve all individuals a s.t. $\mathcal{K} \models \text{a:(Wizard} \sqcap \exists \text{hasPet.Owl)}$

Reasoning Techniques

- **Direct**
  - Specially designed reasoning algorithms
  - Operate on the DL (more or less) directly
- **Indirect**
  - Translate into some equivalent problem in another formalism
  - Solve resulting problem using appropriate technology

Direct Reasoning Techniques

- Two basic classes of algorithm
  - **Model construction**
    - Prove entailment does not hold by constructing model of KB in which axiom/fact is false
    - E.g., tableau algorithms
      - tableau expansion rules used to derive **new ABox facts**
  - **Proof derivation**
    - Prove entailment holds by deriving axiom/fact from KB
    - E.g., structural, completion, rule-based algorithms
      - deduction rules used to derive **new TBox axioms**

Tableau Algorithms

- Currently the most widely used technique
  - Basis for reasoners such as FaCT++, HermiT, Pellet, Racer, …
- Mainly used with more expressive logics (e.g., OWL)
  - Standard technique is to negate premise axiom/fact
  - HyperTableau may also work well with sub-Boolean DLs
- Most effective for schema reasoning
  - Large datasets may necessitate construction of large models
  - Query answering may require each possible answer to be checked
  - Optimisations can limit but not eliminate these problems

Tableau Algorithms

- Transform entailment to KB (un)satisfiability
  - $\mathcal{K} \models a{:}C$ iff $\mathcal{K} \cup \{a{:}(\neg C)\}$ is *not* satisfiable
  - $\mathcal{K} \models C \sqsubseteq D$ iff $\mathcal{K} \cup \{a{:}(C \sqcap \neg D)\}$ is *not* satisfiable (for new a)
- Start with facts explicitly asserted in ABox

  e.g., John:HappyParent, John hasChild Mary
- Use expansion rules to derive new **ABox facts**

  e.g., John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
- Construction fails if obvious contradiction (clash)

  e.g., Mary:Doctor, Mary:$\neg$Doctor

Expansion Rules for $\mathcal{ALC}$

$\sqcap$-rule: if 1. $a : (C_1 \sqcap C_2) \in \mathcal{A}$, and
2. $\{a : C_1, a : C_2\} \not\subseteq \mathcal{A}$
then set $\mathcal{A}_1 = \mathcal{A} \cup \{a : C_1, a : C_2\}$

$\sqcup$-rule: if 1. $a : (C_1 \sqcup C_2) \in \mathcal{A}$, and
2. $\{a : C_1, a : C_2\} \cap \mathcal{A} = \emptyset$
then set $\mathcal{A}_1 = \mathcal{A} \cup \{a : C_1\}$ and $\mathcal{A}_2 = \mathcal{A} \cup \{a : C_2\}$

$\exists$-rule: if 1. $a : (\exists S.C) \in \mathcal{A}$, and
2. there is no b such that $\{\langle a, b\rangle : S, b : C\} \subseteq \mathcal{A}$,
then set $\mathcal{A}_1 = \mathcal{A} \cup \{\langle a, d\rangle : S, d : C\}$, where d is new in $\mathcal{A}$

$\forall$-rule: if 1. $\{a : (\forall S.C), \langle a, b\rangle : S\} \subseteq \mathcal{A}$, and
2. $b : C \notin \mathcal{A}$
then set $\mathcal{A}_1 = \mathcal{A} \cup \{b : C\}$

- some rules are nondeterministic, e.g., $\sqcup$, $\leq$
- implementations use backtracking search

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

Expansion Example

$\mathcal{T} = \{\text{Doctor} \sqsubseteq \text{Person}, \text{Parent} \equiv \text{Person} \sqcap \exists\text{hasChild.Person},$
$\quad \text{HappyParent} \equiv \text{Parent} \sqcap \forall\text{hasChild.(Doctor} \sqcup \exists\text{hasChild.Doctor)}\}$

$\mathcal{A} = \{\text{John:HappyParent, John hasChild Mary}$

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
✘ Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
✘ Mary:Doctor

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)

Expansion Example

$\mathcal{T} = \{\text{Doctor} \sqsubseteq \text{Person}, \text{Parent} \equiv \text{Person} \sqcap \exists\text{hasChild.Person},$
$\quad \text{HappyParent} \equiv \text{Parent} \sqcap \forall\text{hasChild.(Doctor} \sqcup \exists\text{hasChild.Doctor)}\}$

$\mathcal{A} = \{\text{John:HappyParent, John hasChild Mary}$

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary

Mary:$\neg$Doctor

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

John:Person, John:$\exists$hasChild.Person

Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)

John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)

Mary:$\exists$hasChild.Doctor

Expansion Example

$\mathcal{T} = \{\text{Doctor} \sqsubseteq \text{Person}, \text{Parent} \equiv \text{Person} \sqcap \exists\text{hasChild.Person},$
$\quad \text{HappyParent} \equiv \text{Parent} \sqcap \forall\text{hasChild.(Doctor} \sqcup \exists\text{hasChild.Doctor})\}$

$\mathcal{A} = \{\text{John:HappyParent, John hasChild Mary}$

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
Mary:$\exists$hasChild.Doctor
Mary hasChild b, b:Doctor, b:Person

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
Mary:$\exists$hasChild.Doctor
Mary hasChild b, b:Doctor, b:Person
a:Doctor

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary, **Mary:$\forall$hasChild.$\bot$**

$\models$ Mary:Doctor ?

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary, **Mary:$\forall$hasChild.$\bot$**

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary, Mary:$\forall$hasChild.$\bot$

Expansion Example

$\mathcal{T}$ = {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, John hasChild Mary, **Mary:$\forall$hasChild.$\bot$**

$\models$ Mary:Doctor ?

John:HappyParent, John hasChild Mary, Mary:$\forall$hasChild.$\bot$
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
Mary:$\exists$hasChild.Doctor
Mary hasChild b, b:Doctor, b:Person

Expansion Example

$\mathcal{T}$ = {Doctor $\sqsubseteq$ Person, Parent $\equiv$ Person $\sqcap$ $\exists$hasChild.Person,
HappyParent $\equiv$ Parent $\sqcap$ $\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)}

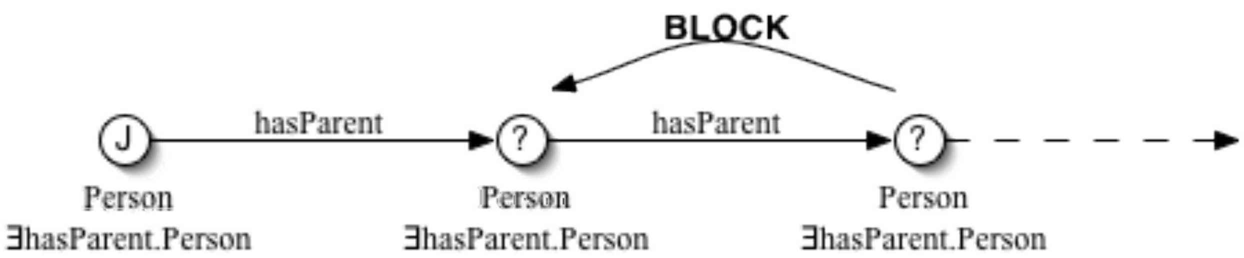
$\mathcal{A}$ = {John:HappyParent, John hasChild Mary, **Mary:$\forall$hasChild.$\bot$**

$\models$ Mary:Doctor ? ✓

John:HappyParent, John hasChild Mary, Mary:$\forall$hasChild.$\bot$
Mary:$\neg$Doctor
John:Parent, John:$\forall$hasChild.(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John:Person, John:$\exists$hasChild.Person
Mary:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
John hasChild a, a:Person, a:(Doctor $\sqcup$ $\exists$hasChild.Doctor)
Mary:$\exists$hasChild.Doctor
Mary hasChild b, b:Doctor, b:Person
✘ b:$\bot$

Termination

- Simplest DLs are naturally terminating
  - Rules produce strictly smaller concepts
- Most DLs require some form of **blocking**
  - E.g., $\{\text{Person} \sqsubseteq \exists\text{hasParent.Person}, \text{John:Person}\}$



- Expressive DLs need more complex blocking

Correctness

A **decision procedure** for KB satisfiability

Will always give an answer, and will always give the ***right*** answer
i.e., it is correct (sound and complete) and terminating

Sound: if clash-free ABox is constructed, then KB is satisfiable

Given fully expanded clash-free ABox, we can trivially construct a model

Complete: if KB is satisfiable, then clash-free ABox is constructed

Given a model, we can use it to guide application of non-deterministic rules

Terminating: the algorithm will always produce an answer

Upper bound on number of new individuals we can create,
so ABox construction will always terminate

Highly Optimised Implementations

- Lazy unfolding
- Simplification and rewriting
  - Absorption: $A \sqcap B \sqsubseteq C \quad \longrightarrow \quad A \sqsubseteq C \sqcup \neg B$
- Detection of tractable fragments ($\mathcal{EL}$)
- Fast semi-decision procedures
  - Told subsumer, model merging, …
- Search optimisations
  - Dependency directed backtracking
- Reuse of previous computations
  - Of (un)satisfiable sets of concepts (conjunctions)
- Heuristics
  - Ordering don't know and don't care non-determinism

Completion Algorithms

- Newer technique, but gaining in popularity
  - Basis for reasoners such as CEL, snorocket, CB, …
- Mainly used with less expressive logics (e.g., OWL 2 EL)
  - Usually restricted to deterministic fragments (e.g., no disjunction)
  - But newer methods may be able to deal with nondeterminism
- Effective with very large schemas
  - Polynomial time algorithms for Horn DLs (such as OWL 2 EL)
  - Finds all subsumption relations in a single computation
- Also effective with very large data sets
  - Polynomial in the size of the data
  - New techniques exploit DB technology for scalability

Completion Algorithms

- Transform KB axioms into simplified form
  - e.g., $C \sqsubseteq \exists R.(C \sqcap D) \quad \leadsto \quad C \sqsubseteq \exists R.A,\ A \sqsubseteq C \sqcap D$
- Use completion rules to derive new **TBox axioms**

$$
\begin{array}{lll}
\text{e.g.,} & & \text{ProudParent} \sqsubseteq \exists\text{hasChild.Doctor}, \\
 & & \text{Doctor} \sqsubseteq \text{Person}, \\
 & & \exists\text{hasChild.Person} \sqsubseteq \text{Parent} \\
 & \leadsto & \text{ProudParent} \sqsubseteq \text{Parent}
\end{array}
$$

- Structural algorithms used with early DLs can be seen as naïve (and typically incomplete) form of completion

Completion Rules for $\mathcal{ELH}$

$$\frac{A \sqsubseteq B \quad B \sqsubseteq C \in \mathcal{O}}{A \sqsubseteq C}$$

$$\frac{A \sqsubseteq B \quad A \sqsubseteq C \quad B \sqcap C \sqsubseteq D \in \mathcal{O}}{A \sqsubseteq D}$$

$$\frac{A \sqsubseteq B \quad B \sqsubseteq \exists r.C \in \mathcal{O}}{A \sqsubseteq \exists r.C}$$

$$\frac{A \sqsubseteq \exists r.B \quad r \sqsubseteq s \in \mathcal{O}}{A \sqsubseteq \exists s.B}$$

$$\frac{A \sqsubseteq \exists r.B \quad B \sqsubseteq C \quad \exists r.C \sqsubseteq D \in \mathcal{O}}{A \sqsubseteq D}$$

Completion Rules for $\mathcal{ELH}$

$$\frac{A \sqsubseteq B \quad B \sqsubseteq C \in \mathcal{O}}{A \sqsubseteq C}$$

$$\frac{A \sqsubseteq B \quad A \sqsubseteq C \quad B \sqcap C \sqsubseteq D \in \mathcal{O}}{A \sqsubseteq D}$$

$$\frac{A \sqsubseteq B \quad B \sqsubseteq \exists r.C \in \mathcal{O}}{A \sqsubseteq \exists r.C}$$

$$\frac{A \sqsubseteq \exists r.B \quad r \sqsubseteq s \in \mathcal{O}}{A \sqsubseteq \exists s.B}$$

$$\boxed{\frac{A \sqsubseteq \exists r.B \quad B \sqsubseteq C \quad \exists r.C \sqsubseteq D \in \mathcal{O}}{A \sqsubseteq D}}$$

Correctness

A **decision procedure** for classification

Will always give an answer, and will always give the ***right*** answer
i.e., it is correct (sound and complete) and terminating

Sound: if $C \sqsubseteq D$ is derived, then KB entails $C \sqsubseteq D$

Completion rules are locally correct (preserve entailments)

Complete: if $C \sqsubseteq D$ is entailed by KB, then $C \sqsubseteq D$ is derived

Completion rules cover all cases

Terminating: the algorithm will always produce an answer

Upper bound on number of axioms of the form $C \sqsubseteq D$ or $C \sqsubseteq \exists r.D$,
so completion will always "saturate"

Query Rewriting

- Basis for systems such as QuOnto, Owlgres and Quill
- Mainly used with less expressive logics (e.g., OWL 2 QL)
  - Usually restricted to deterministic fragments
  - Axioms may also be asymmetric (different restrictions on lhs/rhs)
- Focus is on query answering
  - Usually assume that TBox/schema is small and/or simple
- Effective with very large data sets
  - Rewritings typically produce a Datalog program
  - May even produce union of conjunctive queries (≈ SQL query)
    - Data can be stored/left in relational DB
    - Can delegate query answering to RDBMS

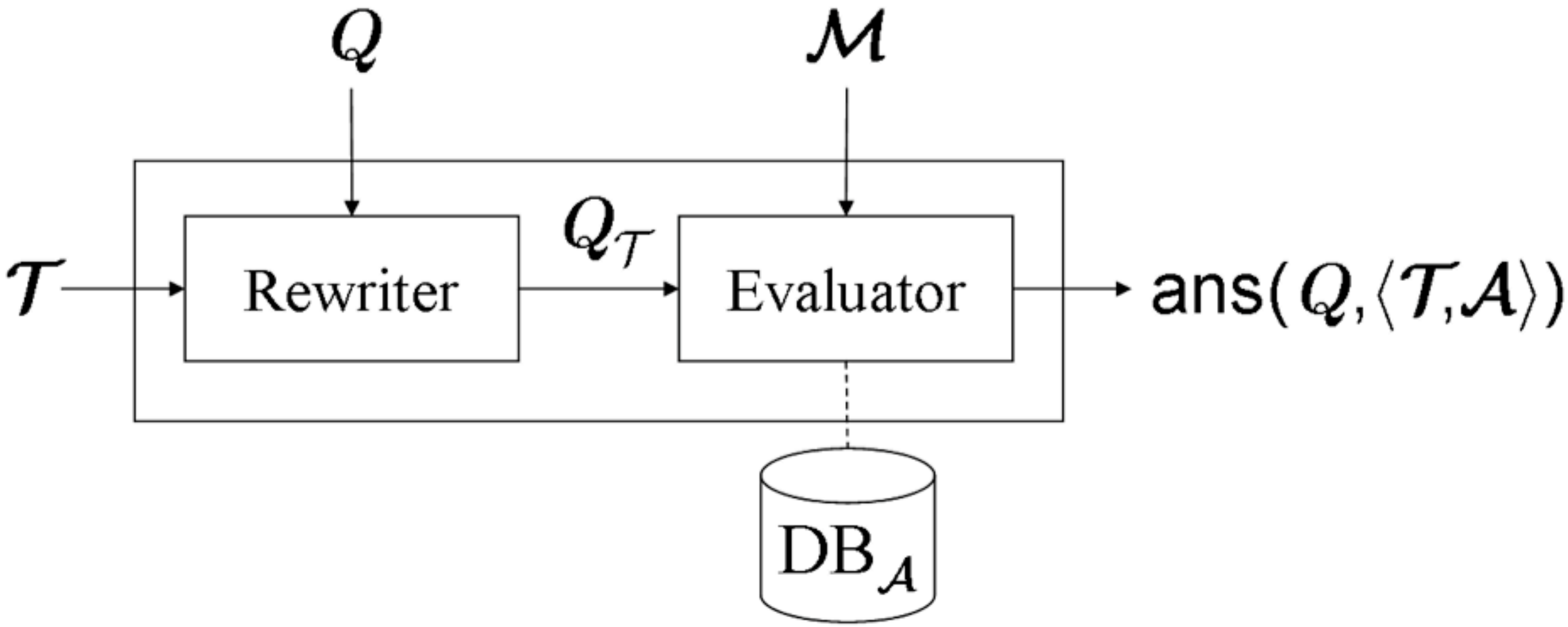
Query Rewriting

- Use KB axioms $\mathcal{T}$ to expand query $\mathcal{Q}$ to query $\mathcal{Q}_\mathcal{T}$

$$\text{e.g.,} \quad \begin{array}{ll} & \text{Professor} \sqsubseteq \text{Teacher}, \\ & \mathcal{Q}(x) \leftarrow \text{Teacher}(x), \\ \rightsquigarrow & \mathcal{Q}_\mathcal{T}(x) \leftarrow \text{Professor}(x) \cup \text{Teacher}(x) \end{array}$$

- Use mappings to evaluate expanded query against DB
  - KB axioms no longer considered (internalised in query)
  - ABox/DB not used in query rewriting
    - Can be used without knowledge of DB contents and/or when access to DB is limited
- Can also use for schema reasoning
  - $C \sqsubseteq D$ iff after adding $a{:}C$ for new individual a, KB $\models a{:}D$

System Architecture



Query Rewriting Example

$$\mathcal{T} = \boxed{\begin{array}{rcl} \text{Teacher} & \sqsubseteq & \exists\text{teaches} \\ \text{Professor} & \sqsubseteq & \text{Teacher} \\ \exists\text{hasTutor}^{-} & \sqsubseteq & \text{Professor} \end{array}}$$

$$Q_0(x) \leftarrow \text{teaches}(x, y)$$

$$\mathcal{M} = \boxed{\begin{array}{l} \text{Professor} \mapsto \texttt{SELECT 1 FROM Professor} \\ \text{hasTutor} \mapsto \texttt{SELECT 1,2 FROM hasTutor} \end{array}}$$

Query Rewriting Example

$$\mathcal{T} = \boxed{\begin{array}{rcl} \text{Teacher} & \sqsubseteq & \exists\text{teaches} \\ \text{Professor} & \sqsubseteq & \text{Teacher} \\ \exists\text{hasTutor}^- & \sqsubseteq & \text{Professor} \end{array}}$$

$$Q_0(x) \leftarrow \text{teaches}(x, y)$$
$$Q_0(x) \leftarrow \text{Teacher}(x)$$
$$Q_0(x) \leftarrow \text{Professor}(x)$$
$$Q_0(x) \leftarrow \text{hasTutor}(y, x)$$

$$\mathcal{M} = \boxed{\begin{array}{l} \text{Professor} \mapsto \texttt{SELECT 1 FROM Professor} \\ \text{hasTutor} \mapsto \texttt{SELECT 1,2 FROM hasTutor} \end{array}}$$

Query Rewriting Example

$$\mathcal{T} = \boxed{\begin{array}{rcl} \text{Teacher} & \sqsubseteq & \exists \text{teaches} \\ \text{Professor} & \sqsubseteq & \text{Teacher} \\ \exists \text{hasTutor}^{-} & \sqsubseteq & \text{Professor} \end{array}}$$

$$Q_0(x) \leftarrow \text{teaches}(x, y)$$
$$Q_0(x) \leftarrow \text{Teacher}(x)$$
$$Q_0(x) \leftarrow \text{Professor}(x)$$
$$Q_0(x) \leftarrow \text{hasTutor}(y, x)$$

$$\mathcal{M} = \boxed{\begin{array}{l} \text{Professor} \mapsto \texttt{SELECT 1 FROM Professor} \\ \text{hasTutor} \mapsto \texttt{SELECT 1,2 FROM hasTutor} \end{array}}$$

$$\mathcal{Q}_{\mathcal{T}} = \boxed{\begin{array}{l} \texttt{SELECT 1 FROM Professor UNION} \\ \texttt{SELECT 2 FROM hasTutor} \end{array}}$$

Query Rewriting Example

$$\mathcal{T} = \boxed{\begin{array}{rcl} \text{Teacher} & \sqsubseteq & \exists\text{teaches} \\ \text{Professor} & \sqsubseteq & \text{Teacher} \\ \exists\text{hasTutor}^{-} & \sqsubseteq & \text{Professor} \end{array}}$$

$$Q_0(x) \leftarrow \text{teaches}(x, y)$$
$$Q_0(x) \leftarrow \text{Teacher}(x)$$
$$Q_0(x) \leftarrow \text{Professor}(x)$$
$$Q_0(x) \leftarrow \text{hasTutor}(y, x)$$

$\mathcal{M}$ =

Professor $\mapsto$ `SELECT 1 FROM Professor`

hasTutor $\mapsto$ `SELECT 1,2 FROM hasTutor`

$\mathcal{Q}_{\mathcal{T}}$ =

```
SELECT 1 FROM Professor UNION
SELECT 2 FROM hasTutor
```

DB =

$\mathit{Professor} = \{\text{Michael}\}$

$\mathit{hasTutor} = \{\langle \text{Rob}, \text{Ian} \rangle, \langle \text{Bruno}, \text{Georg} \rangle\}$

Query Rewriting Example

$$\mathcal{T} = \boxed{\begin{array}{rcl} \text{Teacher} & \sqsubseteq & \exists\text{teaches} \\ \text{Professor} & \sqsubseteq & \text{Teacher} \\ \exists\text{hasTutor}^{-} & \sqsubseteq & \text{Professor} \end{array}}$$

$$Q_0(x) \leftarrow \text{teaches}(x, y)$$
$$Q_0(x) \leftarrow \text{Teacher}(x)$$
$$Q_0(x) \leftarrow \text{Professor}(x)$$
$$Q_0(x) \leftarrow \text{hasTutor}(y, x)$$

$$\mathcal{M} = \boxed{\begin{array}{l} \text{Professor} \mapsto \texttt{SELECT 1 FROM Professor} \\ \text{hasTutor} \mapsto \texttt{SELECT 1,2 FROM hasTutor} \end{array}}$$

$$\mathcal{Q}_{\mathcal{T}} = \boxed{\begin{array}{l} \texttt{SELECT 1 FROM Professor UNION} \\ \texttt{SELECT 2 FROM hasTutor} \end{array}}$$

$$\text{DB} = \boxed{\begin{array}{l} \mathit{Professor} = \{\text{Michael}\} \\ \mathit{hasTutor} = \{\langle \text{Rob, Ian} \rangle, \langle \text{Bruno, Georg} \rangle\} \end{array}}$$

$$\text{ans}(Q_0, \langle \mathcal{T}_0, \mathcal{A}_0 \rangle) = \{\text{Michael, Ian, Georg}\}$$

Correctness

- Rewriting can be shown to be correct

  i.e., $\mathrm{ans}(\mathcal{Q}, \langle \mathcal{T}, \mathcal{A} \rangle) = \mathrm{ans}(\mathcal{Q}_{\mathcal{T}}, \langle \emptyset, \mathcal{A} \rangle)$

- Query answer is correct iff system used to compute $\mathrm{ans}(\mathcal{Q}_{\mathcal{T}}, \langle \emptyset, \mathcal{A} \rangle)$ is correct
  - e.g., if DBMS is sound complete and terminating

Rule-Based Algorithms

- Basis for systems such as Oracle's OWL Prime
  - And widely used to provide *some* OWL support in rule systems
- Mainly used with less expressive logics (e.g., OWL 2 RL)
  - Usually restricted to deterministic and existential-free fragments
    - No disjunction and cannot infer existence of new individuals
  - Syntactic restrictions may also be asymmetric
    - e.g., existentials allowed on lhs of axioms, but not on rhs
- Focus is on query answering
  - Usually assume that TBox/schema is small and/or simple
- Can be effective with large data sets
  - Use rule-extended RDBMS for efficiency

Rule-Based Algorithms

- Rules operate on KB axioms and facts
  - Axioms and facts often in the form of RDF triples
  - e.g., Doctor $\sqsubseteq$ Person, John:Doctor

    $\rightsquigarrow$ <Doctor rdfs:subClassOf Person>, <John rdf:type Doctor>
- Rules **materialise** implied facts (triples) in ABox

  e.g., $\langle ?x \text{ rdf:type } ?c_2 \rangle \leftarrow \langle ?c_1 \text{ rdfs:subClassOf } ?c_2 \rangle \wedge \langle ?x\text{, rdf:type, } ?c_1 \rangle$

  <Doctor rdfs:subClassOf Person>

  <John rdf:type Doctor>

  $\rightsquigarrow$ <John rdf:type Person>
- Rules applied until ABox is **saturated**
  - Query answering then reduces to look-up in saturated Abox
  - Can be delegated to DBMS if saturated ABox stored in DB

Rules for OWL RL (*DLP*)

Table 7. The Semantics of Class Axioms

	If	then	
cax-sco	T(?c_1, rdfs:subClassOf, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)	
cax-eqc1	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_1)	T(?x, rdf:type, ?c_2)	
cax-eqc2	T(?c_1, owl:equivalentClass, ?c_2) T(?x, rdf:type, ?c_2)	T(?x, rdf:type, ?c_1)	
cax-dw	T(?c_1, owl:disjointWith, ?c_2) T(?x, rdf:type, ?c_1) T(?x, rdf:type, ?c_2)	false	
cax-adc	T(?x, rdf:type, owl:AllDisjointClasses) T(?x, owl:members, ?y) LIST[?y, ?c_1, ..., ?c_n] T(?z, rdf:type, ?c_i) T(?z, rdf:type, ?c_j)	false	for each $1 \le i < j \le n$

- There are **many** rules
  - This is only one of 9 tables, most of which are *much* larger

Correctness

- Typically **sound but not complete**
- May be complete for certain kinds of KB + query
  - Implementations based OWL 2 RL rules will be complete w.r.t. atomic facts, i.e., facts of the form

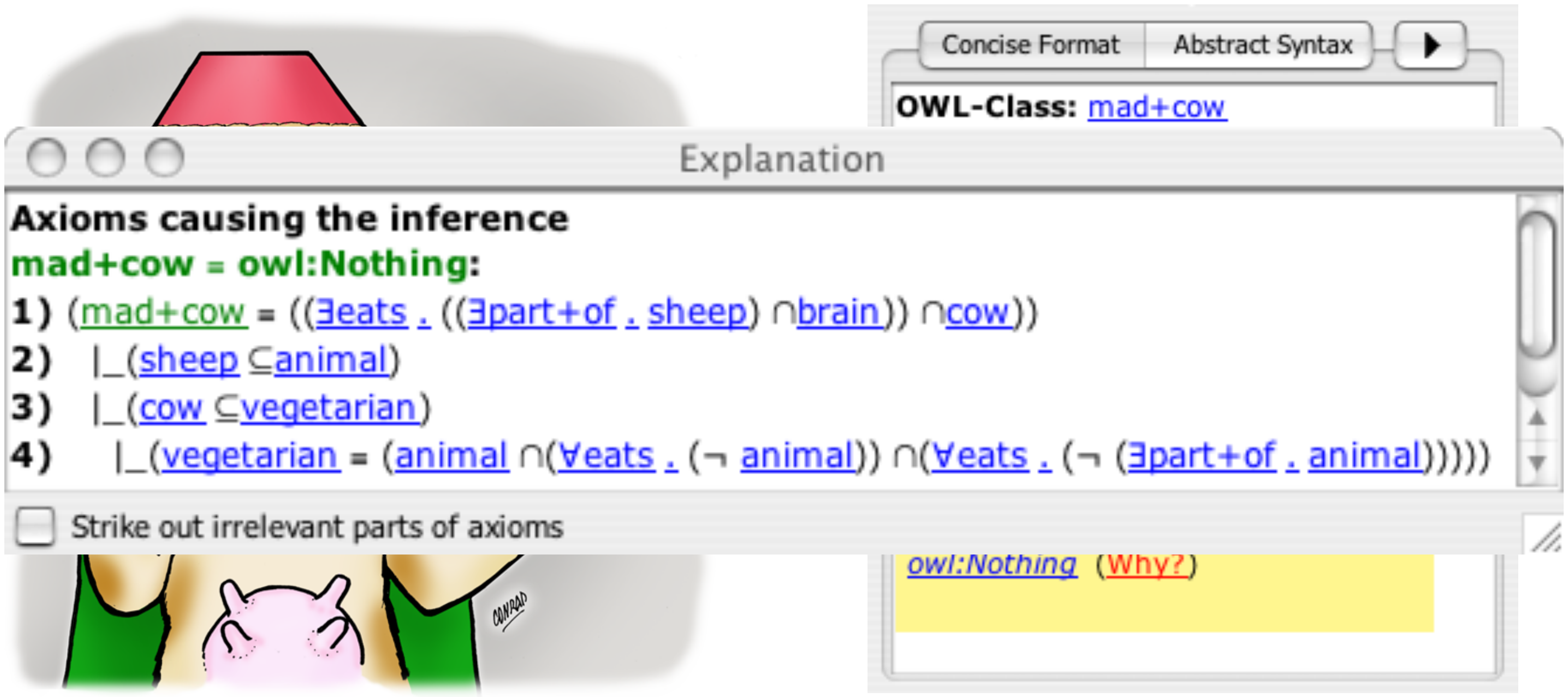
    a:C
    a P b

    where C is a class name and P is a property

Other Reasoning Services

Other Reasoning Services

- Range of new “non-standard” services supporting, e.g.:
  - **Error diagnosis** and repair



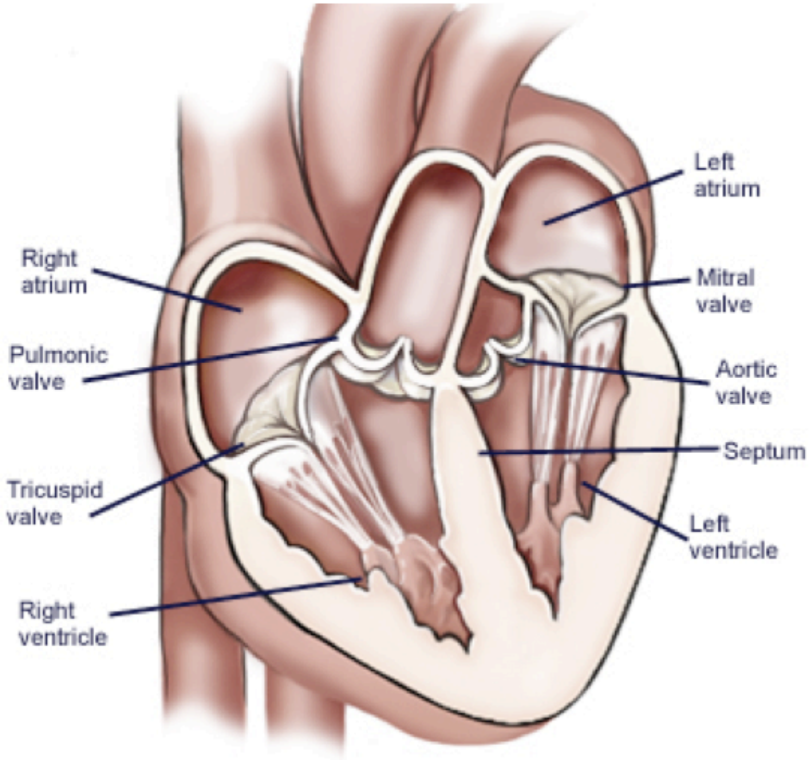
Advanced Reasoning Tasks

- Range of new "non-standard" services supporting, e.g.:
  - **Modular design** and **integration**
    - What is the effect of merging O_2 into O_1?
  - **Module Extraction**
    - Extract a (small) module from O capturing all "relevant" information about some concept or set of concepts
  - **Query and Predicate emptiness**
    - Check if query (or query containing given predicate) is empty for *all* ABoxes
  - **Bottom-up design**
    - Find a (small and specific) concept describing a set of individuals

Recent and Future Work

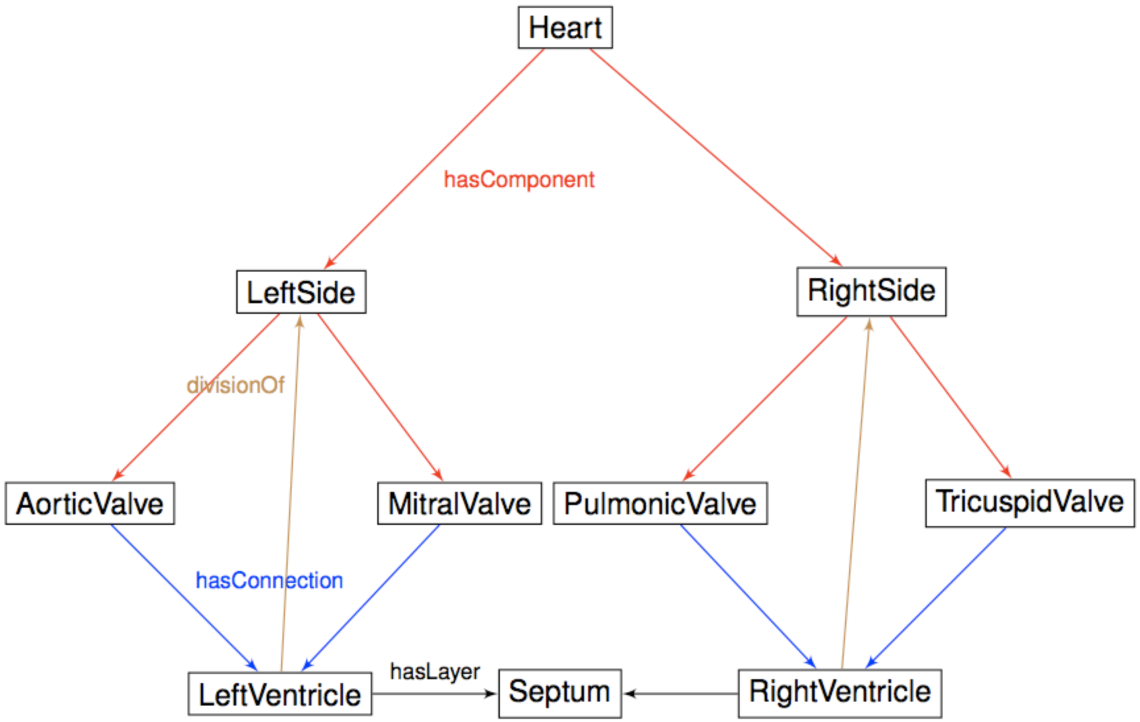
Ontology Languages & Formalisms

- DLs poor for modelling non-tree structures
  - E.g., physically structured objects



Ontology Languages & Formalisms

- DLs poor for modelling non-tree structures
  - E.g., physically structured objects



Ontology Languages & Formalisms

- DLs poor for modelling non-tree structures
  - E.g., physically structured objects
- Description graphs [1] allow for modelling "prototypes"
  - Prototypes resemble small ABoxes
  - Reasoning performance may also be significantly improved
  - Some restrictions needed for decidability
    - E.g., on roles used in TBox and in prototypes

[1] Motik, Cuenca Grau, Horrocks, and Sattler. Representing Structured Objects using Description Graphs. In Proc. of KR 2008.

Ontology Languages & Formalisms

- Integration of (expressive) DLs with DBs
  - Open world semantics can be unintuitive
    - Users may want integrity constraints as well as axioms
  - Reasoning with data can be problematical
    - Scalability & persistence are both issues
  - Solution could be closer integration with DBs [1]
    - Challenge is to find a coherent yet practical semantics

[1] Boris Motik, Ian Horrocks, and Ulrike Sattler. Bridging the Gap Between OWL and Relational Databases. In Proc. of WWW 2007.

New Reasoning Techniques

- New hypertableau calculus [1]
  - Uses more complex hyper-resolution style expansion rules
    - Reduces non-determinism
  - Uses more sophisticated blocking technique
    - Reduces model size
- New HermiT DL reasoner
  - Implements optimised hypertableau algorithm [2]
  - Already outperforms SOTA tableau reasoners

[1] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In Proc. of CADE 2007.

[2] Boris Motik and Ian Horrocks. Individual Reuse in Description Logic Reasoning. In Proc. of IJCAR 2008.

New Reasoning Techniques

- Completion-based decision procedures [1]
  - Use proof search rather than model search
  - Crucial "trick" is to use tableau like techniques to guide and restrict derivations
  - Reasoning time for SNOMED reduced by 2 orders of magnitude

[1] Yevgeny Kazakov. Consequence-Driven Reasoning for Horn SHIQ Ontologies. Proc. of IJCAI 2009 (best paper).

New Reasoning Techniques

- "Combined" decision procedures [1]
  - Combination of materialisation and query rewriting
  - Partial saturation of ABox to deal with existentials
    - adds new "representative" individuals
  - Enhanced query rewriting applied to part-saturated ABox
  - Sound and complete for (at least) OWL 2 EL ontologies
  - Early experiments very encouraging w.r.t. scalability

[1] Carsten Lutz, David Toman, and Frank Wolter. Conjunctive Query Answering in the Description Logic EL using a Relational Database System. Proc. of IJCAI 2009.

New Reasoning Services

- Support for ontology re-use
  - Integrate multiple ontologies [1] and/or Extract (small) modules [2]
  - New reasoning problems arise
    - Conservative extension, safety, ..

[1] Bernardo Cuenca Grau, Yevgeny Kazakov, Ian Horrocks, and Ulrike Sattler. A Logical Framework for Modular Integration of Ontologies. In Proc. of IJCAI 2007.

[2] Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular Reuse of Ontologies: Theory and Practice. JAIR, 31:273-318, 2008.

New Reasoning Services

- Conjunctive query answering
  - Expressive query language for ontologies [1, 2]

$$Q(x, y) \longleftarrow C1(x) \wedge C2(y) \wedge R(x, z) \wedge S(z, y)$$

  - Long-standing open problems
    - E.g., decidability of $\mathcal{SHOIQ}$ conjunctive query answering

[1] Birte Glimm, Ian Horrocks, Carsten Lutz, and Uli Sattler. Conjunctive Query Answering for the Description Logic SHIQ. JAIR, 31:157-204, 2008.

[2] Birte Glimm, Ian Horrocks, and Ulrike Sattler. Unions of Conjunctive Queries in SHOQ. In Proc. of KR 2008.

Summary

- DLs are a family of logic based KR formalisms
  - Useful subsets of First Order Logic
  - Basis for ontology languages such as OWL
- Motivating applications in, e.g., life sciences and semantic web
- Reasoning systems support ontology development & deployment
  - Different reasoning techniques for different applications
  - Robust and scalable reasoning systems available
- Very active research area with many open problems
  - New logics
  - New reasoning tasks
  - New algorithms and implementations
  - …

Resources

- OWL 2
  - Working group http://www.w3.org/2007/OWL/wiki/
  - Language http://www.w3.org/TR/owl2-overview/
  - Systems http://www.w3.org/2007/OWL/wiki/Implementations
- Tools and Systems
  - http://www.cs.man.ac.uk/~sattler/reasoners.html
  - http://protege.stanford.edu/overview/protege-owl.html
- Select bibliography

  - F. Baader, I. Horrocks, and U. Sattler. Description Logics. In *Handbook of Knowledge Representation*. Elsevier, 2007.

    http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/download/2007/BaHS07a.pdf
  - Ian Horrocks. Ontologies and the semantic web. Communications of the ACM, 51(12):58-67, December 2008.
    http://www.comlab.ox.ac.uk/people/ian.horrocks/Publications/download/2008/Horr08a.pdf