DEPARTMENT OF
COMPUTER
SCIENCE

UNIVERSITY OF
OXFORD

Semantic Technologies: Beyond the Semantic Web

Ian Horrocks
Information Systems Group
Department of Computer Science
University of Oxford

The Semantic Web

- Web "invented" by **Tim Berners-Lee** (an Oxford graduate!), then a physicist working at CERN
- His original vision of the Web was much more **ambitious** than the reality of the existing (syntactic) Web:

"… a set of **connected applications** … forming a **consistent logical web of data** … information is given **well-defined meaning**, better enabling computers and people to work in cooperation …"

- This vision of the Web has become known as the **Semantic Web**
- Latest (refined) definition:

  "a web of data that can be processed directly and indirectly by machines"

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Semantic Technologies

- Initial focus was on necessary **underpinning**, including:

Semantic Technologies

- Initial focus was on necessary **underpinning**, including:
  - Languages

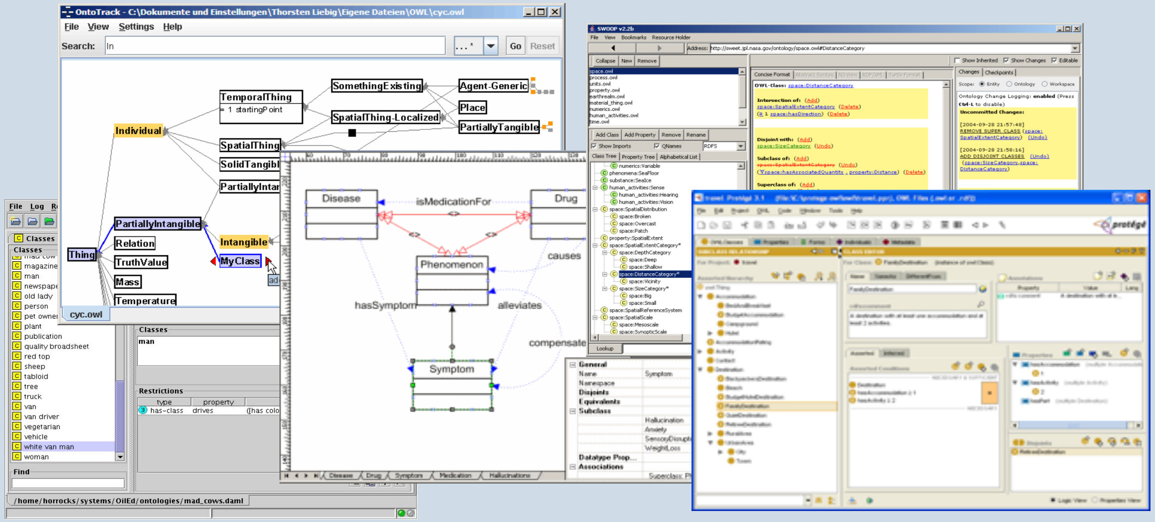
University of Oxford Department of Computer Science
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
Seventh Framework Programme

Semantic Technologies

- Initial focus was on necessary **underpinning**, including:
  - Languages
  - Storage and querying

Semantic Technologies

- Initial focus was on necessary **underpinning**, including:
  - Languages
  - Storage and querying
  - Development tools



UNIVERSITY OF OXFORD — DEPARTMENT OF COMPUTER SCIENCE

Information Systems Group

EPSRC — Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

Semantic Technologies

- Initial focus was on necessary **underpinning**, including:
  - Languages
  - Storage and querying
  - Development tools
- Resulting **robust infrastructure** used in SW applications

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Semantic Technologies

- Initial focus was on necessary **underpinning**, including:
  - Languages
  - Storage and querying
  - Development tools
- Resulting **robust infrastructure** used in SW applications
- Also increasingly used in **“Intelligent Information System”** applications

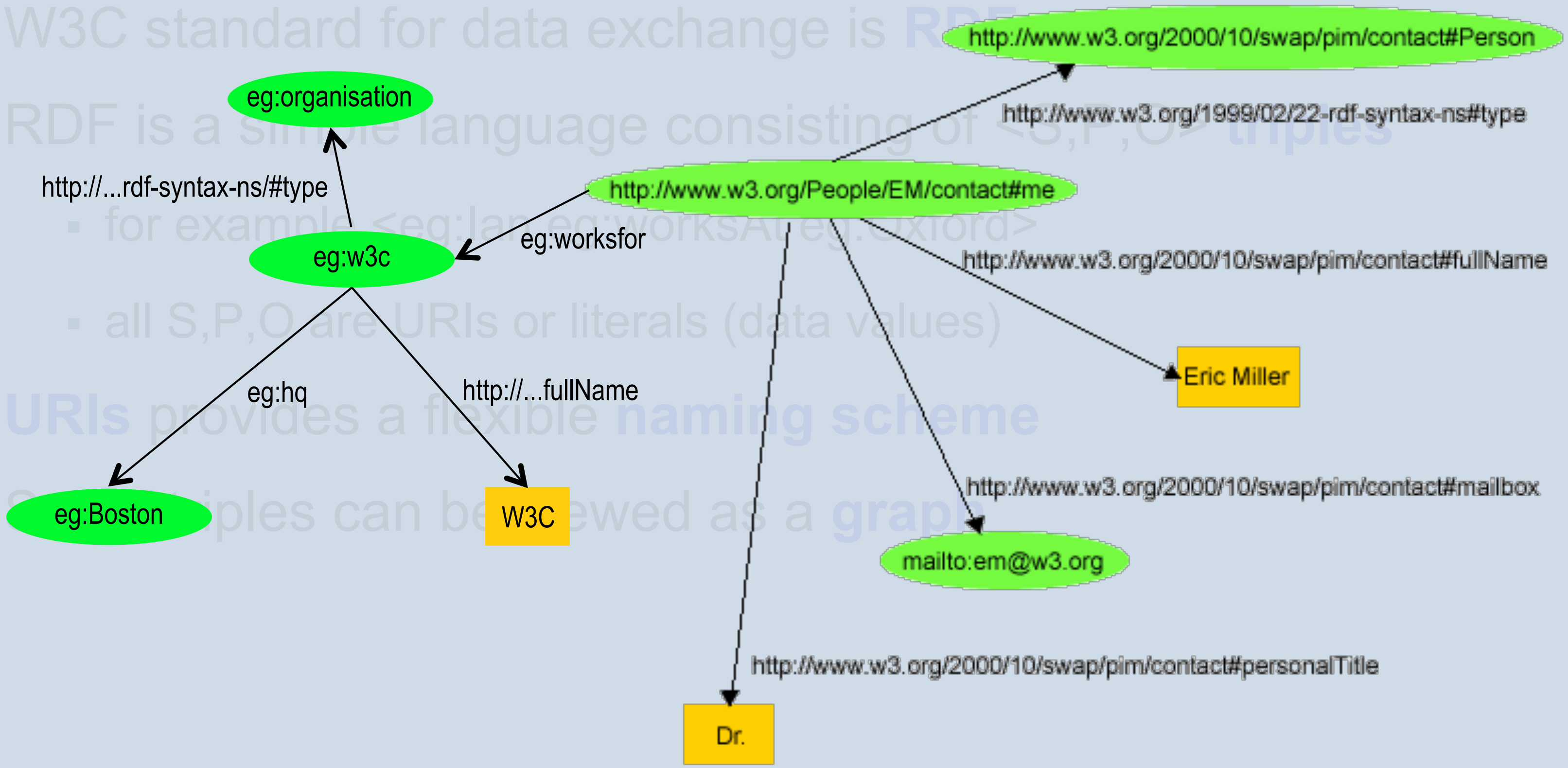
How Does it Work?

❶ Standardised language for exchanging data

- W3C standard for data exchange is **RDF**
- RDF is a simple language consisting of <S P O> **triples**
  - for example <eg:Ian eg:worksAt eg:Oxford>
  - all S,P,O are URIs or literals (data values)
- **URIs** provides a flexible **naming scheme**
- Set of triples can be viewed as a **graph**

How Does it Work?

❶ **Standardised language for exchanging data**



How Does it Work?

RDF

❶ **Standardised language for exchanging data**

- W3C s
- RDF i ... triples
  - for
  - all
- URIs p
- Set of

Triple		
S	**P**	**O**
em1234	rdf:type	Person
em1234	name	“Eric Miller”
em1234	title	“Dr”
em1234	mailbox	mailto:em@w3.org
em1234	worksfor	w3c
w3c	rdf:type	organisation
w3c	hq	Boston
w3c	name	“W3C”
…	…	…

How Does it Work?

❶ Standardised language for exchanging data

- W3C standard for data exchange is RDF
- RD
- URI
- Set

PERSON				
ID	**NAME**	**TITLE**	**MAILBOX**	**WORKSFOR**
em1234	“Eric Miller”	“Dr”	mailto:em@w3.org	w3c
...	...	...	...	...

ORGANISATION		
ID	**NAME**	**HQ**
w3c	“W3C”	Boston
...	...	...

...

How Does it Work?

❷ **Standardised language for exchanging vocabularies/schemas**

- W3C standard for vocabulary/schema exchange is **OWL**
- OWL provides for rich conceptual schemas, aka **ONTOLOGIES**

$$\mathsf{Heart} \sqsubseteq \mathsf{MuscularOrgan} \sqcap \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem}$$

$$\mathsf{HeartDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.\mathsf{Heart}$$

$$\mathsf{VascularDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})$$

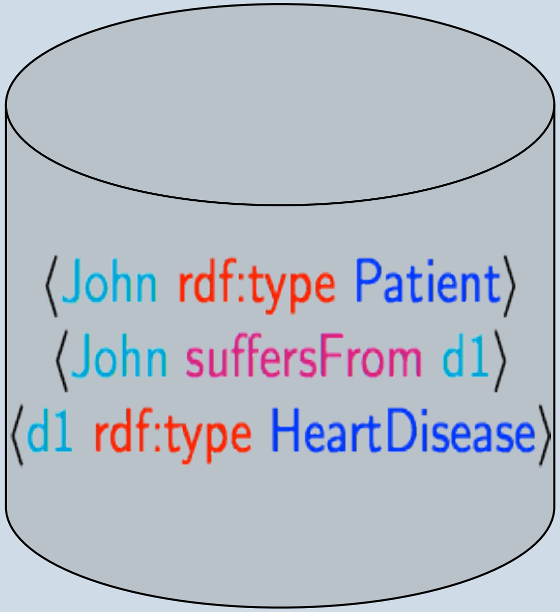
How Does it Work?

❸ **Standardised language for querying ontologies+data**

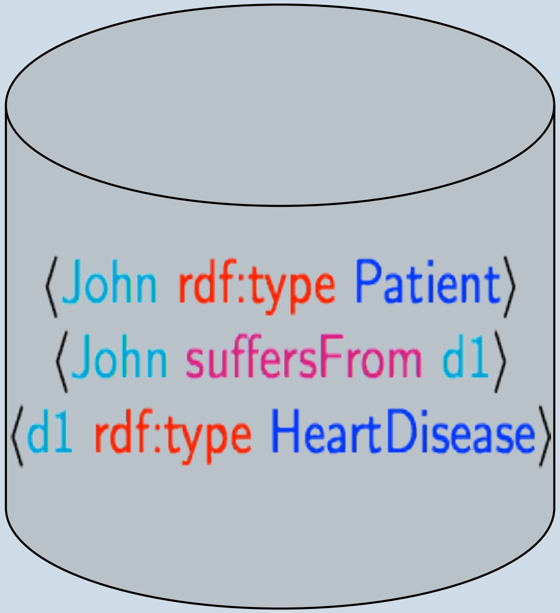
- W3C standard for querying is **SPARQL**
- SPARQL provides a rich query language comparable to SQL
  - ?x worksfor ?y .
    ?y rdf:type organisation .
    ?y hq Boston .
  - Select ?x
    where { ?x worksfor ?y .
    ?y rdf:type organisation .
    ?y hq Boston . }
  - Q(?x) ← worksfor(?x,?y) ∧ organisation(?y) ∧ hq(?y,Boston)

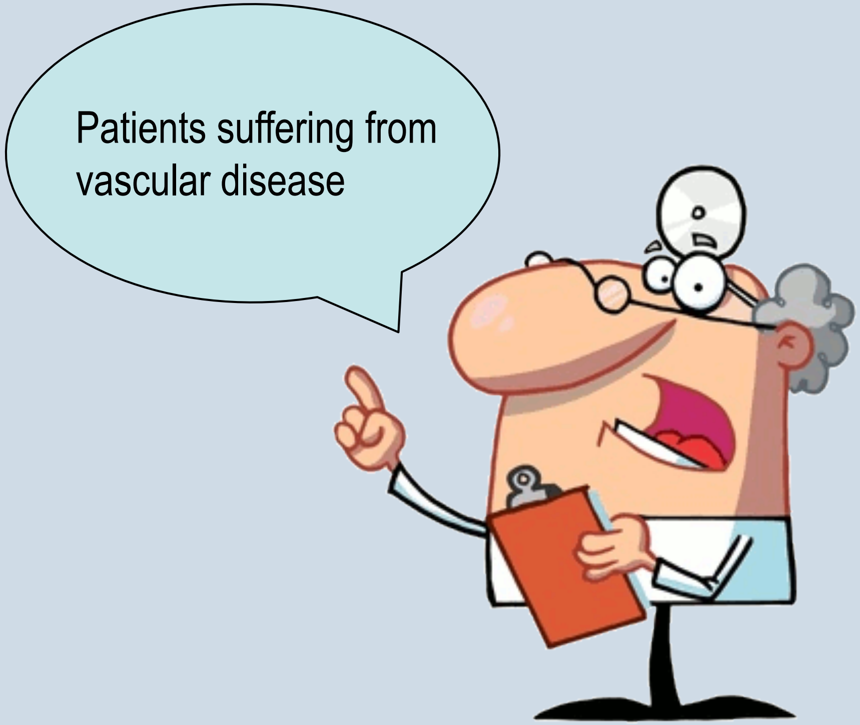
UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

How Does it Work?

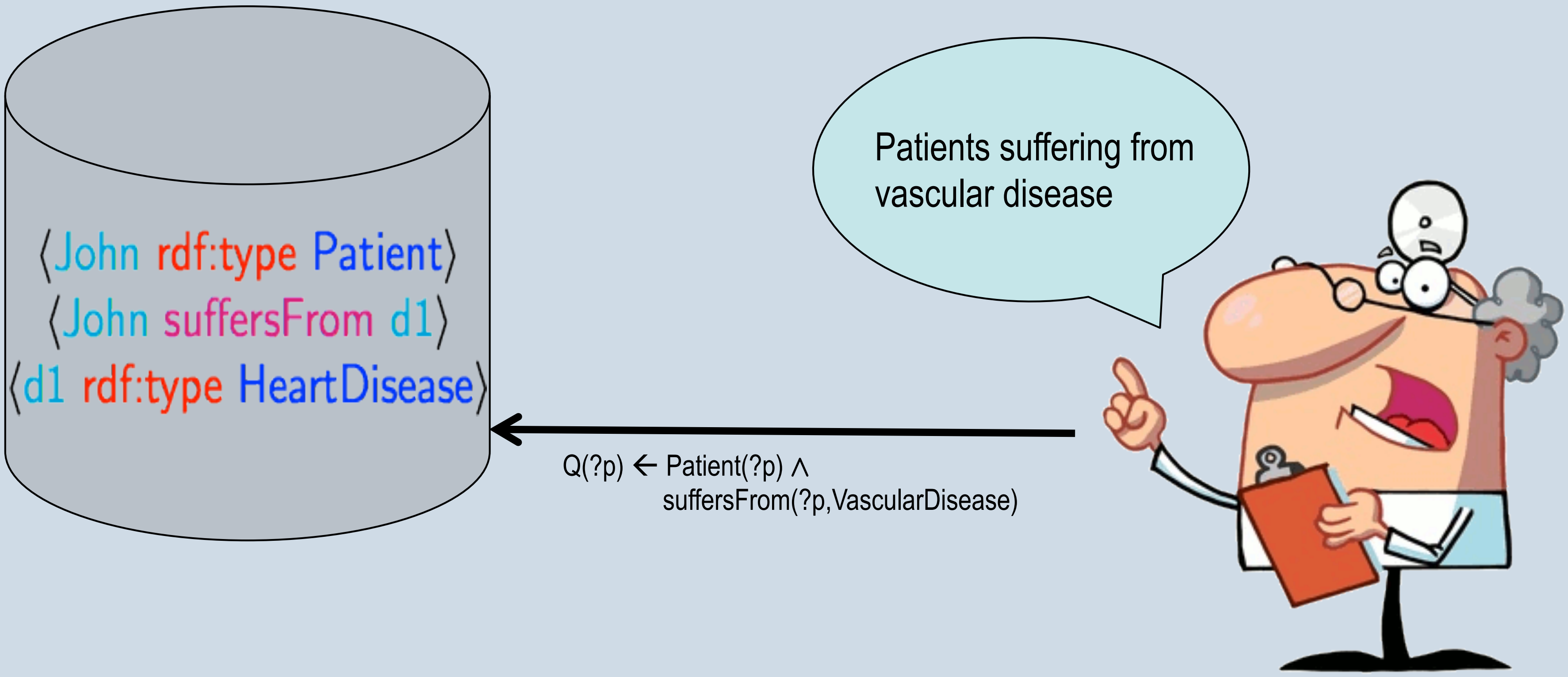


How Does it Work?

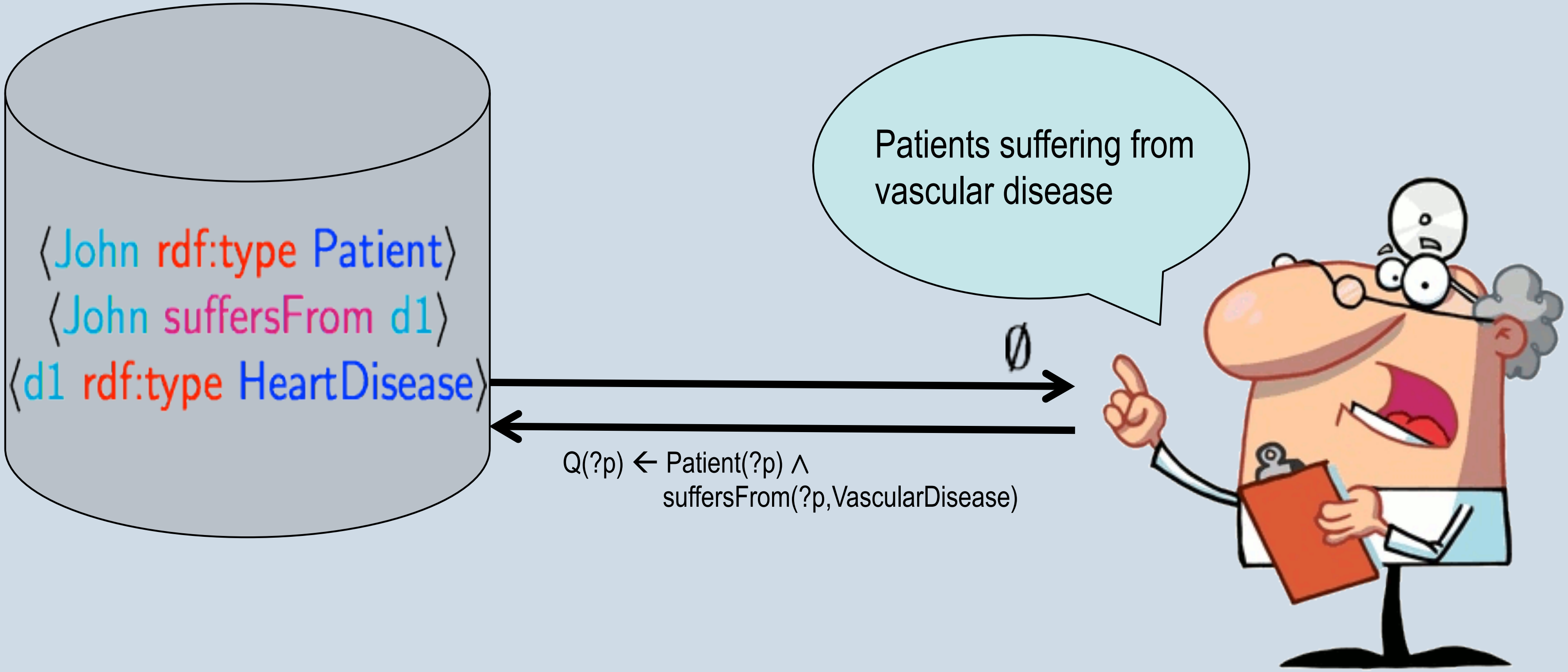




How Does it Work?

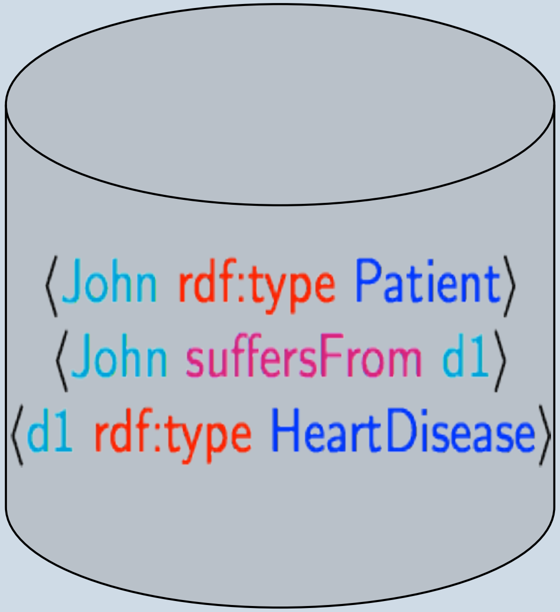


How Does it Work?

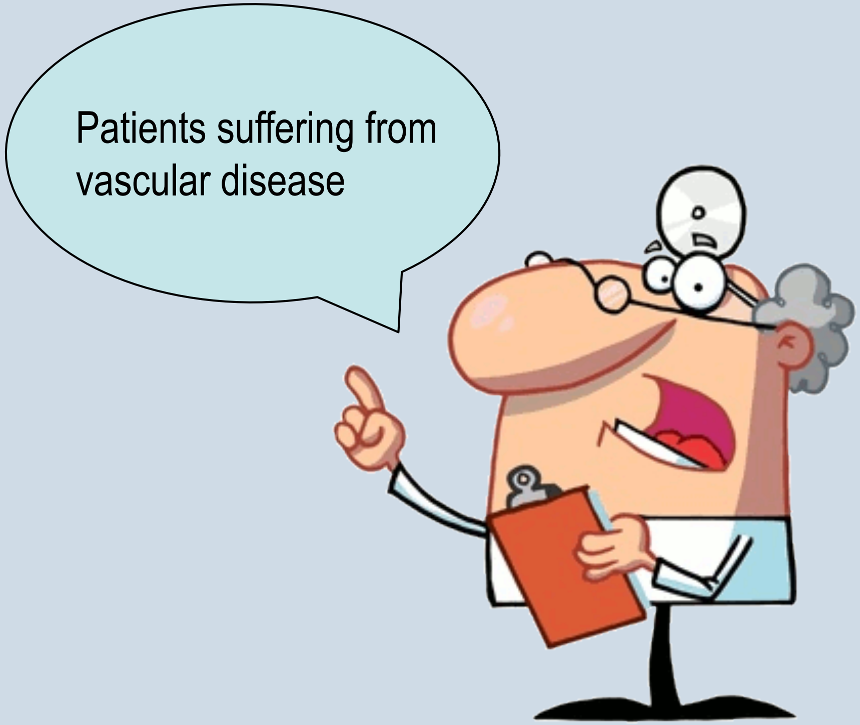


UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

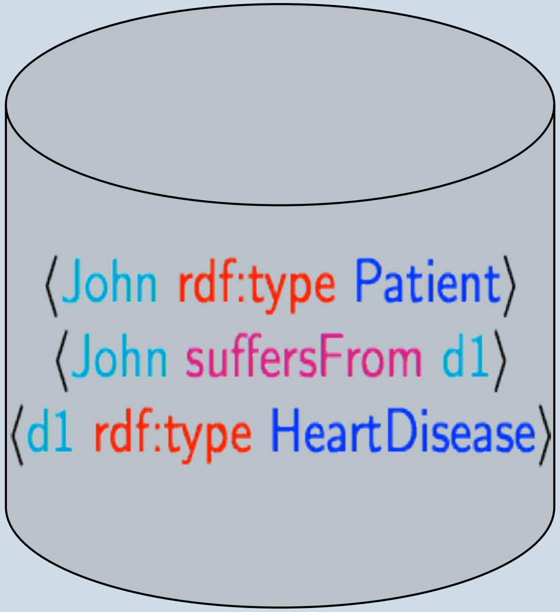
How Does it Work?



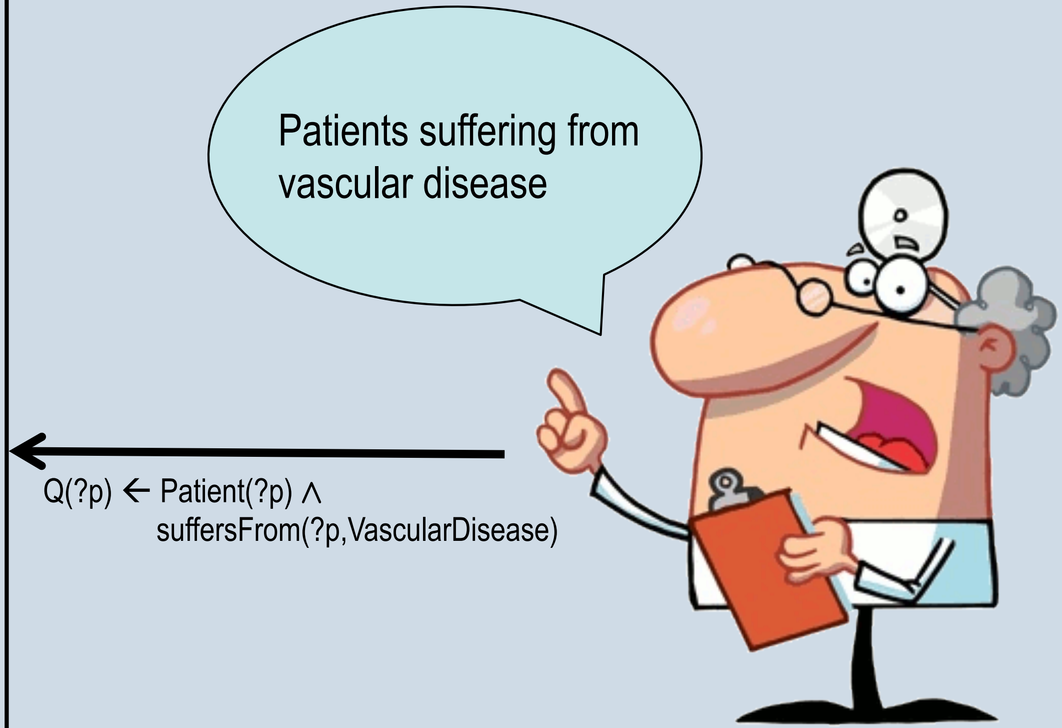
$$
\begin{aligned}
\mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\
&\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\
\mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.\mathsf{Heart} \\
\mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})
\end{aligned}
$$



How Does it Work?

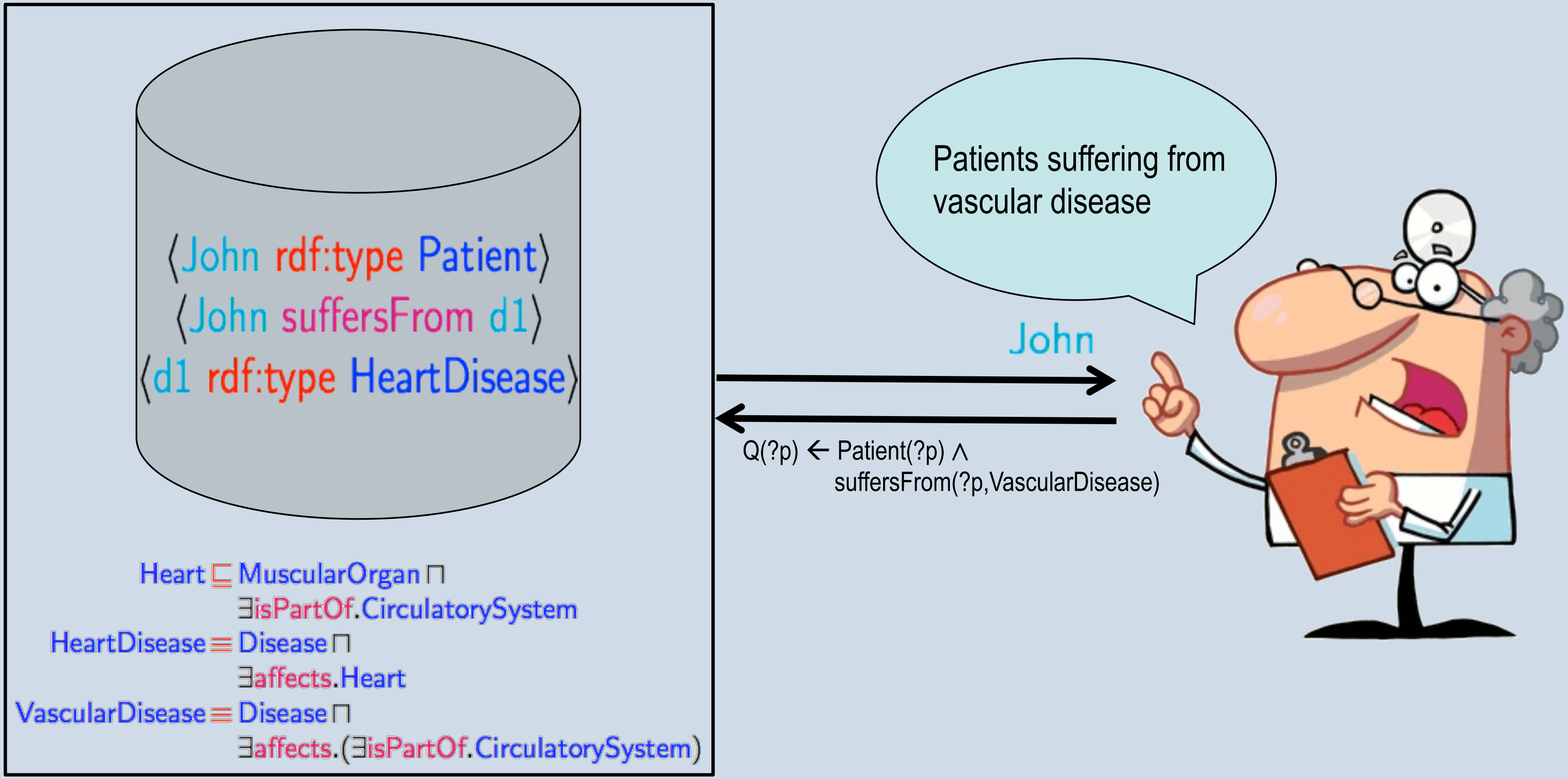


$$\begin{aligned} \mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\ &\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\ \mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\ &\quad \exists \mathsf{affects}.\mathsf{Heart} \\ \mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\ &\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem}) \end{aligned}$$

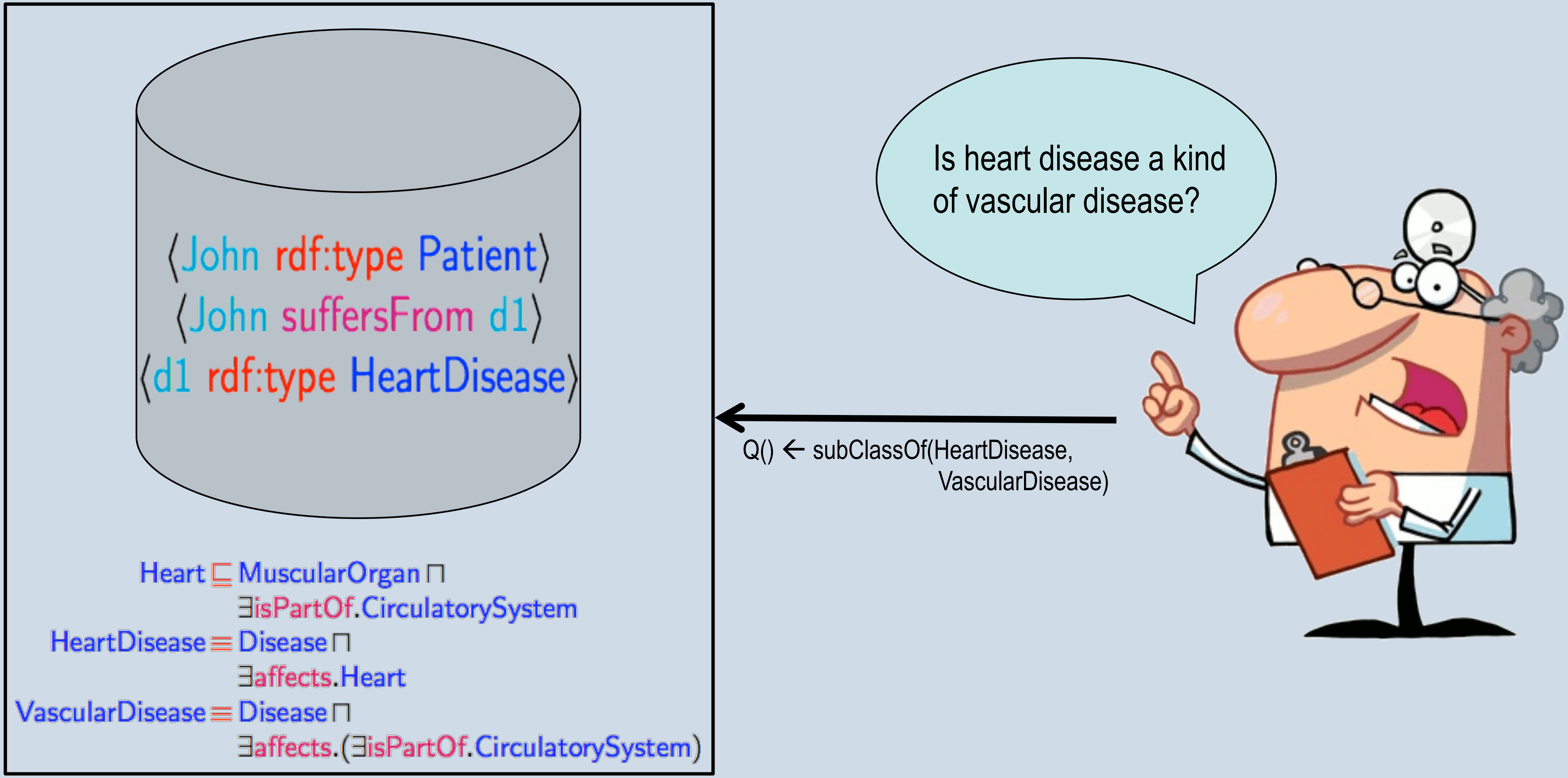


University of Oxford Department of Computer Science
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
Seventh Framework Programme

How Does it Work?



How Does it Work?



How Does it Work?

⟨John rdf:type Patient⟩
⟨John suffersFrom d1⟩
⟨d1 rdf:type HeartDisease⟩

$$\begin{aligned}
\mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\
&\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\
\mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.\mathsf{Heart} \\
\mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})
\end{aligned}$$

Is heart disease a kind of vascular disease?

YES

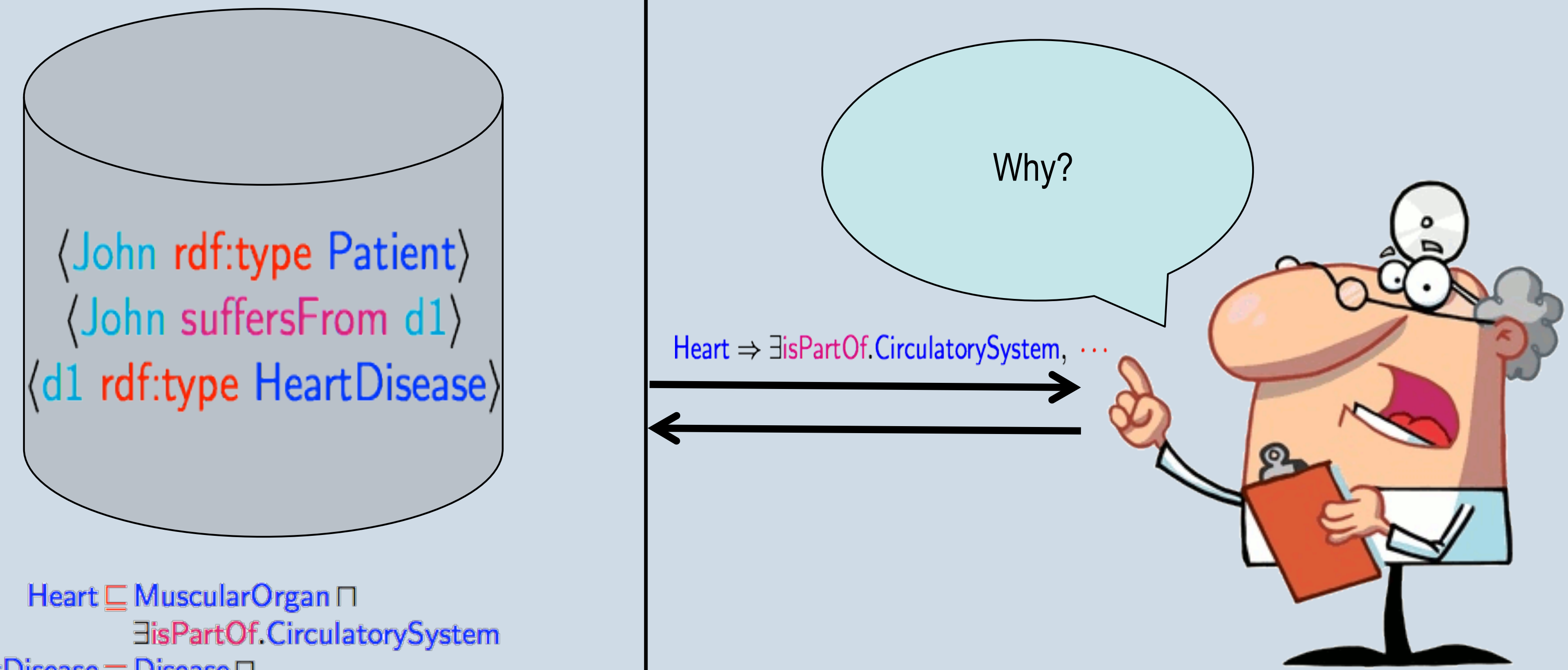
Q() ← subClassOf(HeartDisease, VascularDisease)

How Does it Work?

⟨John rdf:type Patient⟩
⟨John suffersFrom d1⟩
⟨d1 rdf:type HeartDisease⟩

$$\begin{aligned}
\mathsf{Heart} &\sqsubseteq \mathsf{MuscularOrgan} \sqcap \\
&\quad \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem} \\
\mathsf{HeartDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.\mathsf{Heart} \\
\mathsf{VascularDisease} &\equiv \mathsf{Disease} \sqcap \\
&\quad \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})
\end{aligned}$$

How Does it Work?



$$\mathsf{Heart} \sqsubseteq \mathsf{MuscularOrgan} \sqcap \exists \mathsf{isPartOf}.\mathsf{CirculatorySystem}$$

$$\mathsf{HeartDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.\mathsf{Heart}$$

$$\mathsf{VascularDisease} \equiv \mathsf{Disease} \sqcap \exists \mathsf{affects}.(\exists \mathsf{isPartOf}.\mathsf{CirculatorySystem})$$

University of Oxford Department of Computer Science
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
Seventh Framework Programme

Applications: Semantic Web

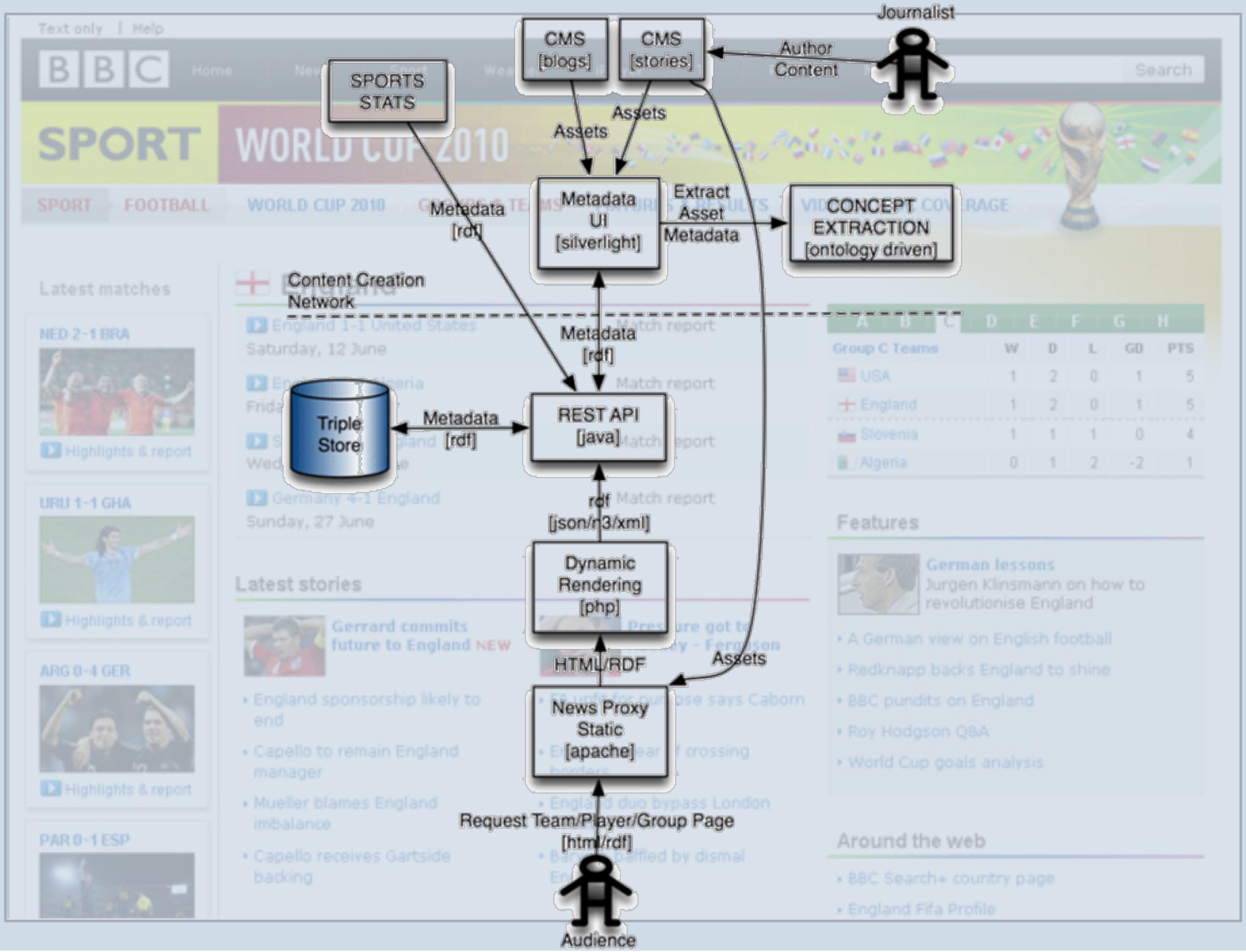
DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF OXFORD

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

Applications: Semantic Web



UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

Applications: Semantic Web

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

Applications: HCLS

- **SNOMED-CT** (Clinical Terms) ontology
  - provides common vocabulary for recording clinical data
  - used in healthcare systems of more than 15 countries, including Australia, Canada, Denmark, Spain, Sweden and the UK
  - “classified and checked for equivalencies” using ontology reasoners
- **OBO foundry** includes more than 100 biological and biomedical ontologies

  “continuous integration server running Elk and/or HermiT 24/7 checking that multiple independently developed ontologies are mutually consistent”
- **Siemens** “actively building OWL based clinical solutions”

Applications: Energy Supply Industry

- **EDF Energy** offer personalised energy saving advice to every customer

- **OWL ontology** used to model relevant environmental factors

- **HermiT reasoner** used to match customer circumstances with relevant pieces of advice

Applications: Intelligent Mobile Platform

- **Samsung** developing Intelligent Moblile Platform to support context-aware applications
- IMP monitors environment via **sensor data** (GPS, compass, accelerometer, ...)
- **OWL ontology** used to model environment and **infer context** (e.g., coffee with friends)
- Applications exploit context to enable more **intelligent behaviour**

Applications: Oil and Gas Industry

- **Statoil** use data to inform production and exploration management

  Large and complex data sets are difficult and time consuming to use

- Semantic technology can **improve access** to relevant data

- Test deployment in EU project Optique

Theory $\rightsquigarrow$ Practice

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD
Information Systems Group
EPSRC
Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Theory $\rightsquigarrow$ Practice

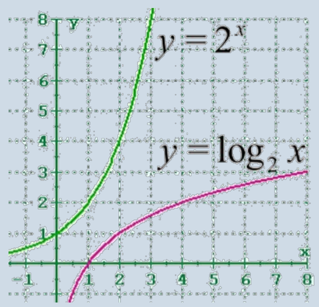
- OWL based on **description logic** $\mathcal{SROIQ}$

Theory $\rightsquigarrow$ Practice

- OWL based on **description logic** $\mathcal{SROIQ}$
- DLs are a family of FOL fragments
  - Clear semantics
  - Well understood computational properties (e.g., decidability, complexity)
  - Simple goal directed reasoning algorithms

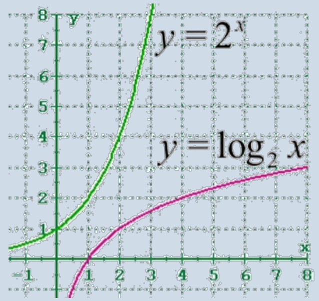
Theory ⇝ Practice

- OWL based on **description logic** $\mathcal{SROIQ}$
- DLs are a family of FOL fragments
  - Clear semantics
  - Well understood computational properties (e.g., decidability, complexity)
  - Simple goal directed reasoning algorithms
- OWL is decidable, but highly **highly intractable**
  - N2ExpTime-comlete combined complexity
  - NP-hard data complexity (-v- logspace for databases)



Theory $\leadsto$ Practice

- OWL based on **description logic $\mathcal{SROIQ}$**
- DLs are a family of FOL fragments
  - Clear semantics
  - Well understood computational properties (e.g., decidability, complexity)
  - Simple goal directed reasoning algorithms
- OWL is decidable, but highly **highly intractable**
  - N2ExpTime-comlete combined complexity
  - NP-hard data complexity (-v- logspace for databases)



How can we provide robustly scalable query answering?

and now:

A Word from our Sponsors

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

What Are Description Logics?

What Are Description Logics?

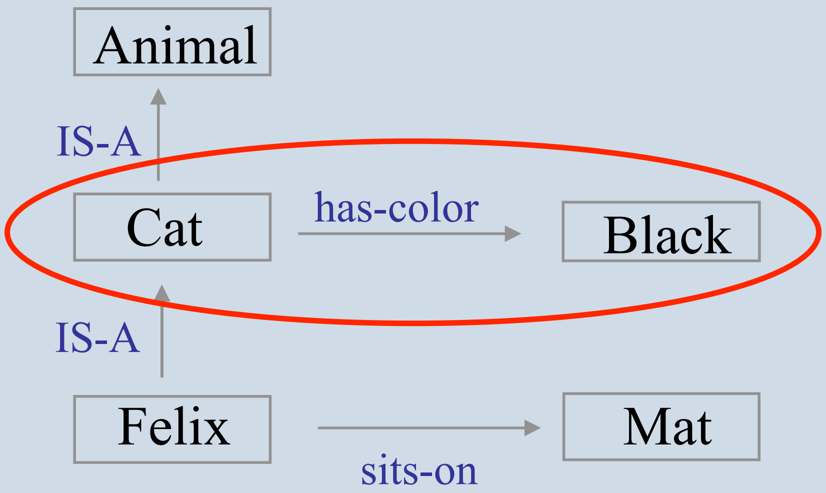
- Decidable fragments of First Order Logic

Thank you for listening

Any questions?

What Are Description Logics?

- A family of logic based Knowledge Representation formalisms
  - Originally descended from semantic networks and KL-ONE
  - Describe domain in terms of concepts (aka classes), roles (aka properties, relationships) and individuals



•[Quillian, 1967]

UNIVERSITY OF OXFORD
DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC
Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

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 C2)
    - 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)
- The basis for widely used ontology languages

DL Syntax

- Signature
  - Concept (aka class) names, e.g., Cat, Animal, Doctor
    - Equivalent to FOL unary predicates
  - Role (aka property) names, e.g., sits-on, hasParent, loves
    - Equivalent to FOL binary predicates
  - Individual names, e.g., Felix, John, Mary, Boston, Italy
    - Equivalent to FOL constants

DL Syntax

- Operators
  - Many kinds available, e.g.,
    - Standard FOL Boolean operators (⊓, ⊔, ¬)
    - Restricted form of quantifiers (∃, ∀)
    - Counting (≥, ≤, =)
    - …

DL Syntax

- Concept expressions, e.g.,
  - $\text{Doctor} \sqcup \text{Lawyer}$
  - $\text{Rich} \sqcap \text{Happy}$
  - $\text{Cat} \sqcap \exists \text{sits-on}.\text{Mat}$
- Equivalent to FOL formulae with one free variable
  - $\text{Doctor}(x) \vee \text{Lawyer}(x)$
  - $\text{Rich}(x) \wedge \text{Happy}(x)$
  - $\text{Cat}(x) \wedge \exists y.(\text{sits-on}(x, y))$

DL Syntax

- Special concepts
  - $\top$ (aka top, Thing, most general concept)
  - $\bot$ (aka bottom, Nothing, inconsistent concept)

  used as abbreviations for
  - $(A \sqcup \neg A)$ for any concept A
  - $(A \sqcap \neg A)$ for any concept A

DL Syntax

- Role expressions, e.g.,
  - loves^-
  - $\text{hasParent} \circ \text{hasBrother}$
- Equivalent to FOL formulae with two free variables
  - $\text{loves}(y, x)$
  - $\exists z.(\text{hasParent}(x, z) \wedge \text{hasBrother}(z, y))$

DL Syntax

- "Schema" Axioms, e.g.,
  - Rich $\sqsubseteq$ ¬Poor (concept inclusion)
  - Cat $\sqcap$ ∃sits-on.Mat $\sqsubseteq$ Happy (concept inclusion)
  - BlackCat $\equiv$ Cat $\sqcap$ ∃hasColour.Black (concept equivalence)
  - sits-on $\sqsubseteq$ touches (role inclusion)
  - Trans(part-of) (transitivity)
- Equivalent to (particular form of) FOL sentence, e.g.,
  - $\forall$x.(Rich(x) $\rightarrow$ ¬Poor(x))
  - $\forall$x.(Cat(x) $\wedge$ ∃y.(sits-on(x,y) $\wedge$ Mat(y)) $\rightarrow$ Happy(x))
  - $\forall$x.(BlackCat(x) $\leftrightarrow$ (Cat(x) $\wedge$ ∃y.(hasColour(x,y) $\wedge$ Black(y)))
  - $\forall$x,y.(sits-on(x,y) $\rightarrow$ touches(x,y))
  - $\forall$x,y,z.((sits-on(x,y) $\wedge$ sits-on(y,z)) $\rightarrow$ sits-on(x,z))

DL Syntax

- "Data" Axioms (aka Assertions or Facts), e.g.,
  - BlackCat(Felix) (concept assertion)
  - Mat(Mat1) (concept assertion)
  - Sits-on(Felix,Mat1) (role assertion)
- Directly equivalent to FOL "ground facts"
  - Formulae with no variables

DL Syntax

- A set of axioms is called a TBox, e.g.:

$\{$Doctor $\sqsubseteq$ Person,
Parent $\equiv$ Person $\sqcap$ $\exists$hasChild.Person,
HappyParent $\equiv$ Parent $\sqcap$ $\forall$hasChild.(Do

- A set of facts is called an ABox, e.g

$\{$HappyParent(John),
hasChild(John,Mary)$\}$

Note

Facts sometimes written
John:HappyParent,
John hasChild Mary,
$\langle$John,Mary$\rangle$:hasChild

- A Knowledge Base (KB) is just a TBox plus an Abox
  - Often written $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$

The DL Family

- Many different DLs, often with "strange" names
  - E.g., $\mathcal{EL}$, $\mathcal{ALC}$, $\mathcal{SHIQ}$
- Particular DL defined by:
  - Concept operators ($\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$, etc.)
  - Role operators ($^{-}$, $\circ$, etc.)
  - Concept axioms ($\sqsubseteq$, $\equiv$, etc.)
  - Role axioms ($\sqsubseteq$, Trans, etc.)

The DL Family

- E.g., $\mathcal{EL}$ is a well known “sub-Boolean” DL
  - Concept operators: $\sqcap$, $\neg$, $\exists$
  - No role operators (only atomic roles)
  - Concept axioms: $\sqsubseteq$, $\equiv$
  - No role axioms
- E.g.:

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

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

The DL Family

- $\mathcal{ALC}$ is the smallest propositionally closed DL
  - Concept operators: $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$
  - No role operators (only atomic roles)
  - Concept axioms: $\sqsubseteq$, $\equiv$
  - No role axioms
- E.g.:

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

The DL Family

- $\mathcal{S}$ used for $\mathcal{ALC}$ extended with (role) transitivity axioms
- Additional letters indicate various extensions, e.g.:
  - $\mathcal{H}$ for role hierarchy (e.g., hasDaughter $\sqsubseteq$ hasChild)
  - $\mathcal{R}$ for role box (e.g., hasParent $\circ$ hasBrother $\sqsubseteq$ hasUncle)
  - $\mathcal{O}$ for nominals/singleton classes (e.g., {Italy})
  - $\mathcal{I}$ for inverse roles (e.g., isChildOf $\equiv$ hasChild$^-$)
  - $\mathcal{N}$ for number restrictions (e.g., $\geqslant 2$hasChild, $\leqslant 3$hasChild)
  - $\mathcal{Q}$ for qualified number restrictions (e.g., $\geqslant 2$hasChild.Doctor)
  - $\mathcal{F}$ for functional number restrictions (e.g., $\leqslant 1$hasMother)
- E.g., $\mathcal{SHIQ}$ = $\mathcal{S}$ + role hierarchy + inverse roles + QNRs

The DL Family

- Numerous other extensions have been investigated
  - Concrete domains (numbers, strings, etc)
  - DL-safe rules (Datalog-like rules)
  - Fixpoints
  - Role value maps
  - Additional role constructors ($\cap$, $\cup$, $\neg$, $\circ$, id, …)
  - Nary (i.e., predicates with arity >2)
  - Temporal
  - Fuzzy
  - Probabilistic
  - Non-monotonic
  - Higher-order
  - …

DL Semantics

Via translaton to FOL, or directly using FO model theory:

Interpretation function $\mathcal{I}$ Interpretation domain $\Delta^{\mathcal{I}}$

Individuals $i^{\mathcal{I}} \in \Delta^{\mathcal{I}}$

John

Mary

Concepts $C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$

Lawyer

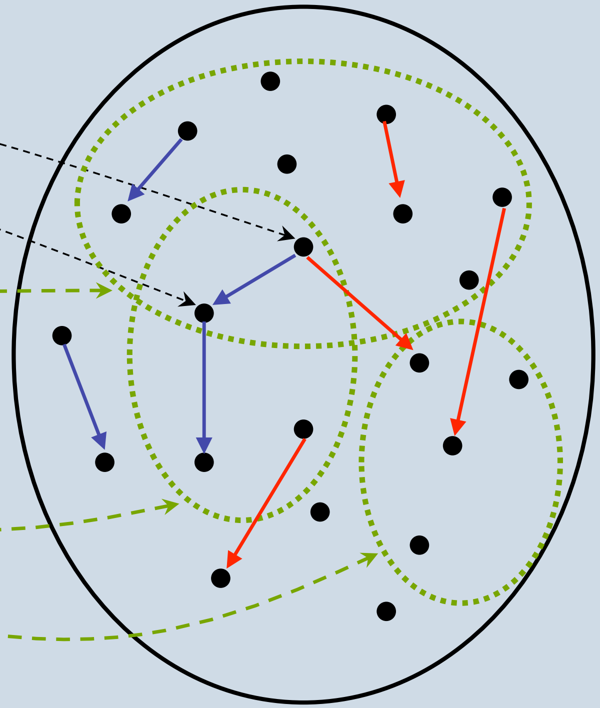
Doctor

Vehicle

Roles $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$

hasChild

owns



DEPARTMENT OF COMPUTER SCIENCE, UNIVERSITY OF OXFORD
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

DL Semantics

- Interpretation function extends to concept expressions in the obvious(ish) way, e.g.:

$$
\begin{aligned}
(C \sqcap D)^{\mathcal{I}} &= C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
(C \sqcup D)^{\mathcal{I}} &= C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
(\neg C)^{\mathcal{I}} &= \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \\
\{x\}^{\mathcal{I}} &= \{x^{\mathcal{I}}\} \\
(\exists R.C)^{\mathcal{I}} &= \{x \mid \exists y. \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\} \\
(\forall R.C)^{\mathcal{I}} &= \{x \mid \forall y. (x, y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\} \\
(\leqslant n R)^{\mathcal{I}} &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^{\mathcal{I}}\} \leqslant n\} \\
(\geqslant n R)^{\mathcal{I}} &= \{x \mid \#\{y \mid \langle x, y \rangle \in R^{\mathcal{I}}\} \geqslant n\}
\end{aligned}
$$

DL Semantics

- Given a model M = $\langle D, \cdot^I \rangle$
  - $M \models C \sqsubseteq D$ iff $C^I \subseteq D^I$
  - $M \models C \equiv D$ iff $C^I = D^I$
  - $M \models C(a)$ iff $a^I \in C^I$
  - $M \models R(a,b)$ iff $\langle a^I, b^I \rangle \in R^I$
  - $M \models \langle \mathcal{T}, \mathcal{A} \rangle$ iff for every axiom $ax \in \mathcal{T} \cup \mathcal{A}$, $M \models ax$

DL Semantics

- Satisfiability and entailment
  - A KB $\mathcal{K}$ is satisfiable iff there exists a model M s.t. $\mathrm{M} \vDash \mathcal{K}$
  - A concept C is satisfiable w.r.t. a KB $\mathcal{K}$ iff there exists a model $\mathrm{M} = \langle \mathrm{D}, \cdot^{\mathrm{I}} \rangle$ s.t. $\mathrm{M} \vDash \mathcal{K}$ and $\mathrm{C}^{\mathrm{I}} \neq \emptyset$
  - A KB $\mathcal{K}$ entails an axiom ax (written $\mathcal{K} \vDash \mathrm{ax}$) iff for every model M of $\mathcal{K}$, $\mathrm{M} \vDash \mathrm{ax}$ (i.e., $\mathrm{M} \vDash \mathcal{K}$ implies $\mathrm{M} \vDash \mathrm{ax}$)

DL Semantics

E.g.,

$\mathcal{T}$ = {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, John hasChild Sally,
Mary:$\neg$Doctor, Mary hasChild Peter, Mary:($\leq$ 1 hasChild)

- ✓ $\mathcal{K} \models$ John:Person ?
- ✓ $\mathcal{K} \models$ Peter:Doctor ?
- ✓ $\mathcal{K} \models$ Mary:HappyParent ?
- What if we add "Mary hasChild Jane" ?

  $\mathcal{K} \models$ Peter = Jane
- What if we add "HappyPerson $\equiv$ Person $\sqcap$ $\exists$hasChild.Doctor" ?

  $\mathcal{K} \models$ HappyPerson $\sqsubseteq$ Parent

DL and FOL

- Most DLs are subsets of C2
  - But reduction to C2 may be (highly) non-trivial
    - Trans(R) naively reduces to $\forall x, y, z. R(x, y) \wedge R(y, z) \rightarrow R(x, z)$
- Why use DL instead of C2?
  - Syntax is succinct and convenient for KR applications
  - Syntactic conformance guarantees being inside C2
    - Even if reduction to C2 is non-obvious
  - Different combinations of constructors can be selected
    - To guarantee decidability
    - To reduce complexity
  - Decidability/complexity landscape mapped out in great detail
    - See http://www.cs.man.ac.uk/~ezolin/dl/

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

$\mathcal{ALC}$ ::= ⊥ | A | ¬C | $C \wedge D$ | $C \vee D$ | ∃$R.C$ | ∀$R.C$

Concept constructors:

- [ ] $\mathcal{F}$ – functionality[2]: (≤1 R)
- [x] $\mathcal{N}$ – (unqualified) number restrictions: (≥n R), (≤n R)
- [ ] $\mathcal{Q}$ – qualified number restrictions: (≥n $R.C$), (≤n $R.C$)
- [x] $\mathcal{O}$ – nominals: {a} or {a_1,...,a_n} ("one-of" constructor)
- [ ] μ – least fixpoint operator: $\mu X.C$
- [ ] $R \subseteq S$ – role-value-maps
- [ ] $f = g$ – agreement of functional role chains ("same-as")

Role constructors: (trans) (reg)

- [x] $\mathcal{I}$ – role inverses: R^{-}
- [ ] ∩ – role intersection[3]: $R \cap S$
- [ ] ∪ – role union: $R \cup S$
- [ ] ¬ – role complement: full
- [ ] o – role chain (composition): RoS
- [ ] * – reflexive-transitive closure[4]: R^*
- [ ] id – concept identity: $id(C)$
- Forbid complex roles[5] in number restrictions[6]

TBox is *internalized* in extensions of $\mathcal{ALCIO}$, see [76, Lemma 4.12], [54, 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)

Role axioms (RBox): (OWL-Lite) (OWL-DL) (OWL 1.1)

- [x] $\mathcal{S}$ – Role transitivity: Trans(R)
- [x] $\mathcal{H}$ – Role hierarchy: $R \subseteq S$
- [ ] $\mathcal{R}$ – Complex role inclusions: Ro$S \subseteq R$, Ro$S \subseteq S$
- [ ] s – some additional features

(Reset) You have selected the Description Logic: $\mathcal{SHOIN}$

Complexity of reasoning problems[7]

Reasoning problem	Complexity[8]	Comments and references
Concept satisfiability	**NExpTime-complete**	• Hardness of even $\mathcal{ALCFIO}$ is proved in [76, Corollary 4.13]. In that paper, the result is formulated for $\mathcal{ALCQIO}$, but only number restrictions of the form (≤1R) are used in the proof. • A different proof of the NExpTime-hardness for $\mathcal{ALCFIO}$ is given in [54] (even with 1 nominal, and role inverses not used in number restrictions). • Upper bound for $\mathcal{SHOIQ}$ is proved in [77, 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 $\mathcal{SHOIQ}$. • **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 [46]. • **Remark:** recently [47] it was observed that, in many cases, one can use transitive roles in number restrictions – and still have a decidable logic! So the above notion of a *simple* role could be substantially extended.
ABox consistency	**NExpTime-complete**	By reduction to concept satisfiability problem in presence of nominals shown in [69, Theorem 3.7].

DEPARTMENT OF COMPUTER SCIENCE — UNIVERSITY OF OXFORD

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

Complexity Measures

- Taxonomic complexity
  - Measured w.r.t. total size of “schema” axioms
- Data complexity
  - Measured w.r.t. total size of “data” facts
- Query complexity
  - Measured w.r.t. size of query
- Combined complexity
  - Measured w.r.t. total size of KB (plus query if appropriate)

Complexity Classes

- LogSpace, PTime, NP, PSpace, ExpTime, etc
  - worst case for a given problem w.r.t. a given parameter
  - X-hard means at-least this hard (could be harder);
    in X means no harder than this (could be easier);
    X-complete means both hard and in, i.e., exactly this hard
    - e.g., $\mathcal{SROIQ}$ KB satisfiability is 2NExpTime-complete w.r.t. combined complexity and NP-hard w.r.t. data complexity
- Note that:
  - this is for the worst case, not a typical case
  - complexity of problem means we can never devise a more efficient (in the worst case) algorithm
  - complexity of algorithm may, however, be even higher (in the worst case)

DLs and Ontology Languages

- W3C's OWL 2 (like OWL, DAML+OIL & OIL) based on DL
  - OWL 2 based on $\mathcal{SROIQ}$, i.e., $\mathcal{ALC}$ extended with transitive roles, a role box nominals, inverse roles and qualified number restrictions
    - OWL 2 EL based on $\mathcal{EL}$
    - OWL 2 QL based on DL-Lite
    - OWL 2 EL based on $\mathcal{DLP}$
  - OWL was based on $\mathcal{SHOIN}$
    - only simple role hierarchy, and unqualified NRs

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

Class/Concept Constructors

OWL Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human $\sqcap$ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor $\sqcup$ Lawyer
complementOf	$\neg C$	$\neg$Male
oneOf	$\{x_1\} \sqcup \ldots \sqcup \{x_n\}$	{john} $\sqcup$ {mary}
allValuesFrom	$\forall P.C$	$\forall$hasChild.Doctor
someValuesFrom	$\exists P.C$	$\exists$hasChild.Lawyer
maxCardinality	$\leqslant n P$	$\leqslant$1hasChild
minCardinality	$\geqslant n P$	$\geqslant$2hasChild

Ontology Axioms

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

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

- An Ontology is *usually* considered to be a TBox
  - but an OWL ontology is a mixed set of TBox and ABox axioms

Other OWL Features

- XSD datatypes and (in OWL 2) facets, e.g.,
  - integer, string and (in OWL 2) real, float, decimal, datetime, …
  - minExclusive, maxExclusive, length, …
  - PropertyAssertion(hasAge Meg "17"^^xsd:integer)
  - DatatypeRestriction(xsd:integer xsd:minInclusive "5"^^xsd:integer xsd:maxExclusive "10"^^xsd:integer)

  These are equivalent to (a limited form of) **DL concrete domains**

- Keys
  - E.g., HasKey(Vehicle Country LicensePlate)
    - Country + License Plate is a unique identifier for vehicles

  This is equivalent to (a limited form of) **DL safe rules**

Obvious Database Analogy

- Ontology axioms analogous to DB **schema**
  - Schema describes structure of and constraints on data
- Ontology facts analogous to DB **data**
  - Instantiates schema
  - Consistent with schema constraints
- But there are also important differences…

Obvious Database Analogy

Database:

- Closed world assumption (**CWA**)
  - Missing information treated as false
- Unique name assumption (**UNA**)
  - Each individual has a single, unique name
- Schema behaves as **constraints** on structure of data
  - Define legal database states

Ontology:

- Open world assumption (**OWA**)
  - Missing information treated as unknown
- **No UNA**
  - Individuals may have more than one name
- Ontology axioms behave like **implications** (inference rules)
  - Entail implicit information

Database -v- Ontology

E.g., given the following **ontology/schema**:

$\text{HogwartsStudent} \equiv \text{Student} \sqcap \exists\, \text{attendsSchool.Hogwarts}$

$\text{HogwartsStudent} \sqsubseteq \forall \text{hasPet.(Owl or Cat or Toad)}$

$\text{hasPet} \equiv \text{isPetOf}^{-}$ (i.e., hasPet inverse of isPetOf)

$\exists \text{hasPet}.\top \sqsubseteq \text{Human}$ (i.e., domain of hasPet is Human)

$\text{Phoenix} \sqsubseteq \forall \text{isPetOf.Wizard}$ (i.e., only Wizards have Phoenix pets)

$\text{Muggle} \sqsubseteq \neg \text{Wizard}$ (i.e., Muggles and Wizards are disjoint)

Database -v- Ontology

And the following **facts/data**:

HarryPotter: Wizard
DracoMalfoy: Wizard
HarryPotter hasFriend RonWeasley
HarryPotter hasFriend HermioneGranger
HarryPotter hasPet Hedwig

Query: Is Draco Malfoy a friend of HarryPotter?

- DB: No
- Ontology: Don’t Know

  OWA (didn’t say Draco was not Harry’s friend)

Database -v- Ontology

And the following **facts/data**:

HarryPotter: Wizard
DracoMalfoy: Wizard
HarryPotter hasFriend RonWeasley
HarryPotter hasFriend HermioneGranger
HarryPotter hasPet Hedwig

Query: How many friends does Harry Potter have?

- DB: 2
- Ontology: at least 1

  No UNA (Ron and Hermione may be 2 names for same person)

Database -v- Ontology

And the following **facts/data**:

HarryPotter: Wizard
DracoMalfoy: Wizard
HarryPotter hasFriend RonWeasley
HarryPotter hasFriend HermioneGranger
HarryPotter hasPet Hedwig

➔ **RonWeasley ≠ HermioneGranger**

Query: How many friends does Harry Potter have?

- DB: 2
- Ontology: at least 2

  OWA (Harry may have more friends we didn't mention yet)

Database -v- Ontology

And the following **facts/data**:

HarryPotter: Wizard
DracoMalfoy: Wizard
HarryPotter hasFriend RonWeasley
HarryPotter hasFriend HermioneGranger
HarryPotter hasPet Hedwig
RonWeasley ≠ HermioneGranger

➔ **HarryPotter: ∀hasFriend.{RonWeasley} ⊔ {HermioneGranger}**

Query: How many friends does Harry Potter have?

- DB: 2
- Ontology: 2!

Database -v- Ontology

Inserting new facts/data:

Dumbledore: Wizard
Fawkes: Phoenix
Fawkes isPetOf Dumbledore

$\exists$hasPet.$\top \sqsubseteq$ Human
Phoenix $\sqsubseteq \forall$isPetOf.Wizard

What is the response from DBMS?

- Update rejected: **constraint violation**

  Domain of hasPet is Human; Dumbledore is not Human (CWA)

What is the response from Ontology reasoner?

- **Infer** that Dumbledore is Human (domain restriction)
- Also infer that Dumbledore is a Wizard (only a Wizard can have a pheonix as a pet)

DB Query Answering

- Schema plays **no role**
  - Data must explicitly satisfy schema constraints
- Query answering amounts to **model checking**
  - I.e., a “look-up” against the data
- Can be very **efficiently implemented**
  - Worst case complexity is low (logspace) w.r.t. size of data

Ontology Query Answering

- Ontology axioms play a powerful and **crucial role**
  - Answer may include implicitly derived facts
  - Can answer conceptual as well as extensional queries
    - E.g., Can a Muggle have a Phoenix for a pet?
- Query answering amounts to **theorem proving**
  - I.e., logical entailment
- May have very **high worst case complexity**
  - E.g., for OWL, NP-hard w.r.t. size of data
    (upper bound is an open problem)
  - Implementations may still behave well in typical cases
  - Fragments/profiles may have much better complexity

Ontology Based Information Systems

- Analogous to **relational database management systems**
  - Ontology ≈ schema; instances ≈ data
- Some important (**dis**)**advantages**
  - \+ (Relatively) easy to maintain and update schema
    - Schema plus data are integrated in a logical theory
  - \+ Query answers reflect both schema and data
  - \+ Can deal with incomplete information
  - \+ Able to answer both intensional and extensional queries
  - Semantics can seem counter-intuitive, particularly w.r.t. data
    - Open -v- closed world; axioms -v- constraints
  - Query answering (logical entailment) may be much more difficult
    - Can lead to scalability problems with expressive logics

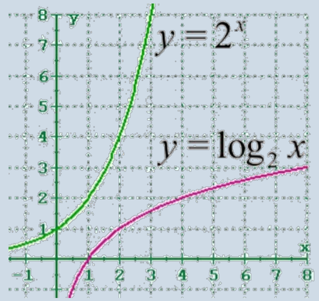
Ontology Based Information Systems

- Analogous to **relational** … **ement systems**
  - Ontology ≈ sche…
- Some important (…
  - \+ (Relatively) …
    - Schema…
  - \+ Query ans…
  - \+ Can deal w…
  - \+ Able to ans… …ries
  - Semantics ca… …w.r.t. data
    - Open -v- clo…
  - Query answering (… …much more difficult
    - Can lead to scalability p… …pressive logics

Back to our Scheduled Program

Theory $\rightsquigarrow$ Practice

- OWL based on **description logic $\mathcal{SROIQ}$**
- DLs are a family of FOL fragments
  - Clear semantics
  - Well understood computational properties (e.g., decidability, complexity)
  - Simple goal directed reasoning algorithms
- OWL is decidable, but highly **highly intractable**
  - N2ExpTime-comlete combined complexity
  - NP-hard data complexity (-v- logspace for databases)



How can we provide robustly scalable query answering?

Various Approaches — Different Tradeoffs

❶ Use full power of OWL and a complete reasoner:

✓ Well-suited for modeling complex domains

✓ Reliable answers

✗ High worst-case complexity

✗ Scalability problems for large ontologies & datasets

Complete OWL reasoners:

- E.g., FaCT++, **HermiT**, Pellet, ...
- Based on (hyper)tableau (model construction) theorem provers
- Highly optimised implementations effective on many ontologies, but not robust and unlikely to scale to large data sets

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff
    KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB $\models$ HeartDisease $\sqsubseteq$ VascularDisease iff KB $\cup$ {x:(HeartDisease $\sqcap$ $\neg$VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease $\sqcap$ $\neg$VascularDisease

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB $\models$ HeartDisease $\sqsubseteq$ VascularDisease iff KB $\cup$ {x:(HeartDisease $\sqcap$ $\neg$VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease $\sqcap$ $\neg$VascularDisease

x : HeartDisease

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff
    KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : HeartDisease

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : HeartDisease

x : Disease

x : ∃affects.Heart

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : HeartDisease

x : Disease

x : ∃affects.Heart

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff
    KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : HeartDisease

x : Disease

x : ∃affects.Heart

(x, y) : affects

y : Heart

y : MuscularOrgan

y : ∃isPartOf.CirculatorySystem

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE

Information Systems Group

EPSRC Engineering and Physical Sciences Research Council

SEVENTH FRAMEWORK PROGRAMME

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : HeartDisease

x : Disease

x : ∃affects.Heart

(x, y) : affects

y : Heart

y : MuscularOrgan

y : ∃isPartOf.CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : HeartDisease

x : Disease

x : ∃affects.Heart

(x, y) : affects

y : Heart

y : MuscularOrgan

y : ∃isPartOf.CirculatorySystem

(y, z) : isPartOf

z : CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease

x : ¬VascularDisease

x : HeartDisease

x : Disease

x : ∃affects.Heart

(x, y) : affects

y : Heart

y : MuscularOrgan

y : ∃isPartOf.CirculatorySystem

(y, z) : isPartOf

z : CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease x : ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB $\models$ HeartDisease $\sqsubseteq$ VascularDisease iff KB $\cup$ {x:(HeartDisease $\sqcap$ $\neg$VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease $\sqcap$ $\neg$VascularDisease
x : HeartDisease
x : Disease
x : $\exists$affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : $\exists$isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : $\neg$VascularDisease
x : $\neg$Disease $\sqcup$ $\neg\exists$affects.($\exists$isPartOf.CirculatorySystem)

DEPARTMENT OF COMPUTER SCIENCE
UNIVERSITY OF OXFORD
Information Systems Group
EPSRC
Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB $\models$ HeartDisease $\sqsubseteq$ VascularDisease iff KB $\cup$ {x:(HeartDisease $\sqcap$ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease $\sqcap$ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease $\sqcup$ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬Disease

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬Disease

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB $\models$ HeartDisease $\sqsubseteq$ VascularDisease iff KB $\cup$ {x:(HeartDisease $\sqcap$ $\neg$VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease $\sqcap$ $\neg$VascularDisease
x : HeartDisease
x : Disease
x : $\exists$affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : $\exists$isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : $\neg$VascularDisease
x : $\neg$Disease $\sqcup$ $\neg\exists$affects.($\exists$isPartOf.CirculatorySystem)
x : $\neg\exists$affects.($\exists$isPartOf.CirculatorySystem)

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ∀affects.(∀isPartOf.¬CirculatorySystem)

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff
    KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔
¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ∀affects.(∀isPartOf.¬CirculatorySystem)

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ∀affects.(∀isPartOf.¬CirculatorySystem)
y : ∀isPartOf.¬CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ∀affects.(∀isPartOf.¬CirculatorySystem)
y : ∀isPartOf.¬CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB $\models$ HeartDisease $\sqsubseteq$ VascularDisease iff KB $\cup$ {x:(HeartDisease $\sqcap$ $\neg$VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease $\sqcap$ $\neg$VascularDisease
x : HeartDisease
x : Disease
x : $\exists$affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : $\exists$isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : $\neg$VascularDisease
x : $\neg$Disease $\sqcup$ $\neg\exists$affects.($\exists$isPartOf.CirculatorySystem)
x : $\neg\exists$affects.($\exists$isPartOf.CirculatorySystem)
x : $\forall$affects.($\forall$isPartOf.$\neg$CirculatorySystem)
y : $\forall$isPartOf.$\neg$CirculatorySystem
z : $\neg$CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ∀affects.(∀isPartOf.¬CirculatorySystem)
y : ∀isPartOf.¬CirculatorySystem
z : ¬CirculatorySystem

(Hyper)tableau — How Does It Work?

Standard technique based on (hyper-) **tableau**

- Reasoning tasks reducible to (un)**satisfiability**
  - E.g., KB ⊨ HeartDisease ⊑ VascularDisease iff KB ∪ {x:(HeartDisease ⊓ ¬VascularDisease)} is *not* satisfiable
- Algorithm tries to construct (an abstraction of) a model

x : HeartDisease ⊓ ¬VascularDisease
x : HeartDisease
x : Disease
x : ∃affects.Heart
(x, y) : affects
y : Heart
y : MuscularOrgan
y : ∃isPartOf.CirculatorySystem
(y, z) : isPartOf
z : CirculatorySystem

x : ¬VascularDisease
x : ¬Disease ⊔ ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ¬∃affects.(∃isPartOf.CirculatorySystem)
x : ∀affects.(∀isPartOf.¬CirculatorySystem)
y : ∀isPartOf.¬CirculatorySystem
z : ¬CirculatorySystem

Note similarity to chase!

Various Approaches — Different Tradeoffs

❷ Use a suitable "profile" and specialised reasoner:

OWL 2 defines language subsets, aka **profiles** that can be "more simply and/or efficiently implemented"

- **OWL 2 EL**
  - Based on $\mathcal{EL}^{++}$
  - PTime-complete for combined and data complexity
- **OWL 2 QL**
  - Based on DL-Lite
  - AC^0 data complexity (same as DBs)
- **OWL 2 RL**
  - Based on "**Description Logic Programs**" ($\approx$ DL $\cap$ LP)
  - PTime-complete for combined and data complexity

Various Approaches — Different Tradeoffs

❷ Use a suitable “profile” and specialised reasoner:

✓ Tractable query answering

✓ Reliable answers (for inputs in the profile)

✗ Restricted expressivity of the ontology language

✗ Reasoners reject inputs outside profile

OWL 2 EL ontology reasoners:

- E.g., CEL, ELK, ...
- Based on “consequence based” (deduction) theorem provers
- Target HCLS applications where many ontologies are (mainly) in the EL profile

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Consequence Based — How Does It Work?

- Normalise ontology axioms to standard form:

$$A \sqsubseteq B \quad A \sqcap B \sqsubseteq C \quad A \sqsubseteq \exists R.B \quad \exists R.B \sqsubseteq C$$

- Saturate using inference rules (for $\mathcal{EL}$):

$$\frac{A \sqsubseteq B \quad B \sqsubseteq C}{A \sqsubseteq C} \qquad \frac{A \sqsubseteq B \quad A \sqsubseteq C \quad B \sqcap C \sqsubseteq D}{A \sqsubseteq D}$$

$$\frac{A \sqsubseteq \exists R.B \quad B \sqsubseteq C \quad \exists R.C \sqsubseteq D}{A \sqsubseteq D}$$

- Extension to $\mathcal{EL}^{++}$ requires (many) more rules

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$

$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$

$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$

$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$

$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$

$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$
$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$

Consequence Based — Example

$$\text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site}.\text{Organ}$$
$$\text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site}.\text{Heart}$$
$$\text{Heart} \sqsubseteq \text{Organ}$$

$$\text{OrganTransplant} \sqsubseteq \text{Transplant}$$
$$\text{OrganTransplant} \sqsubseteq \exists \text{site}.\text{Organ}$$
$$\exists \text{site}.\text{Organ} \sqsubseteq \text{SO}$$
$$\text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$

$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$

$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$

$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$

$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$

$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$
$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$
$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$
$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$
$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$$
$$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$$
$$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$
$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$
$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$$
$$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$$
$$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$
$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$
$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$$
$$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$$
$$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$

$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$

$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$

$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$

$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$

$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$

$$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$$

$$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$$

$$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$$

$$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$$

$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

Consequence Based — Example

$$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$$
$$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

$$\frac{A \sqsubseteq \exists R.B \quad B \sqsubseteq C \quad \exists R.C \sqsubseteq D}{A \sqsubseteq D}$$

$$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$$
$$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$$
$$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$$
$$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$$
$$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$$
$$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$$
$$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$$

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Consequence Based — Example

$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$

$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$

$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$

$$\frac{A \sqsubseteq \exists R.B \quad B \sqsubseteq C \quad \exists R.C \sqsubseteq D}{A \sqsubseteq D}$$

$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$

$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$

$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$

$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$

$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$

$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$

$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$

$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$

$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$

$\mathsf{HeartTransplant} \sqsubseteq \mathsf{SO}$

Consequence Based — Example

$$\text{OrganTransplant} \equiv \text{Transplant} \sqcap \exists \text{site}.\text{Organ}$$
$$\text{HeartTransplant} \equiv \text{Transplant} \sqcap \exists \text{site}.\text{Heart}$$
$$\text{Heart} \sqsubseteq \text{Organ}$$

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

$$\text{OrganTransplant} \sqsubseteq \text{Transplant}$$
$$\text{OrganTransplant} \sqsubseteq \exists \text{site}.\text{Organ}$$
$$\exists \text{site}.\text{Organ} \sqsubseteq \text{SO}$$
$$\text{Transplant} \sqcap \text{SO} \sqsubseteq \text{OrganTransplant}$$
$$\text{HeartTransplant} \sqsubseteq \text{Transplant}$$
$$\text{HeartTransplant} \sqsubseteq \exists \text{site}.\text{Heart}$$
$$\exists \text{site}.\text{Heart} \sqsubseteq \text{SH}$$
$$\text{Transplant} \sqcap \text{SH} \sqsubseteq \text{HeartTransplant}$$
$$\text{Heart} \sqsubseteq \text{Organ}$$

$$\text{HeartTransplant} \sqsubseteq \text{SO}$$

Consequence Based — Example

$\mathsf{OrganTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Organ}$

$\mathsf{HeartTransplant} \equiv \mathsf{Transplant} \sqcap \exists \mathsf{site}.\mathsf{Heart}$

$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$

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

$\mathsf{OrganTransplant} \sqsubseteq \mathsf{Transplant}$

$\mathsf{OrganTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Organ}$

$\exists \mathsf{site}.\mathsf{Organ} \sqsubseteq \mathsf{SO}$

$\mathsf{Transplant} \sqcap \mathsf{SO} \sqsubseteq \mathsf{OrganTransplant}$

$\mathsf{HeartTransplant} \sqsubseteq \mathsf{Transplant}$

$\mathsf{HeartTransplant} \sqsubseteq \exists \mathsf{site}.\mathsf{Heart}$

$\exists \mathsf{site}.\mathsf{Heart} \sqsubseteq \mathsf{SH}$

$\mathsf{Transplant} \sqcap \mathsf{SH} \sqsubseteq \mathsf{HeartTransplant}$

$\mathsf{Heart} \sqsubseteq \mathsf{Organ}$

$\mathsf{HeartTransplant} \sqsubseteq \mathsf{SO}$

$\mathsf{HeartTransplant} \sqsubseteq \mathsf{OrganTransplant}$

Schema Reasoning — Solved Problem?

	SNOMED CT	GALEN	FMA	GO
Logic	$\mathcal{EL}$	$\mathcal{EL}$	$\mathcal{EL}$	$\mathcal{EL}$
#classes	315,489	23,136	78,977	19,468
#properties	58	950	7	1
#axioms	430,844	36,547	121,712	28,897
$\#\sqsubseteq$	$> 10^{11}$	$> 10^{8}$	$> 10^{9}$	$> 10^{8}$
ELK (1 worker)	13.15	1.33	0.44	0.20
ELK (4 workers)	5.02	0.77	0.39	0.19

	Plant Anat.	SWEET-P	NCI-2	DOLCE-P
Logic	$\mathcal{SHIF}$	$\mathcal{SHOIN}$	$\mathcal{ALCH}$	$\mathcal{SHOIN}$
#classes	19,145	1,728	70,576	118
#properties	82	145	189	264
#axioms	35,770	2,419	100,304	265
$\#\sqsubseteq$	$> 10^{8}$	$> 10^{6}$	$> 10^{9}$	$> 10^{4}$
HermiT	11.2	11.2	—	105.1
Pellet	87.2	—	172.0	105.1
FaCT++	22.9	0.2	60.7	—

Schema Reasoning — Solved Problem?

- Full expressive power may be needed to model, e.g.:
  - *non-viral pneumonia* (negation)
  - *infectious pneumonia* is caused by a *virus* or a *bacterium* (disjunction)
  - *double pneumonia* occurs in two *lungs* (cardinalities)
  - *groin* has a part that is part of the *abdomen*, and has a part that is part of the *leg* (inverse properties)
- Single non-EL axiom may incur massive performance penalty

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

MORe Modular Reasoner

- Integrates powerful (slower) and weaker (faster) reasoners
- Exploits module extraction techniques to identify subset of ontology that can be completely classified using fast reasoner.
- Slower reasoner performs as few computations as possible
- Bulk of computation delegated to faster reasoner
- Current prototype integrates **HermiT** and **ELK** [1]

[1] Armas Romero, Cuenca Grau, and Horrocks. Modular Combination of Reasoners for Ontology Classification. In Proc. of ISWC 2012 (to appear).

MORe Modular Reasoner

Ontology	$\|\mathcal{O} \setminus \mathcal{O}_{\mathcal{L}}\|$	$\|\Sigma^{\mathcal{L}}\|$	$\|\mathcal{M}_{[\mathcal{O},\overline{\Sigma^{\mathcal{L}}}]}\|$	Classif. time (seconds)			
				HermiT	MORe		
					total	HermiT	ELK
GO	0	100%	0%	7.1	2.2 (↓69.0%)	0	0.1
Gazeteer	0	100%	0%	838.1	28.2 (↓96.6%)	0	15.6
NCI	65	94.9%	15.4%	84.1	28.6 (↓66.0%)	15.8	3.3
Protein	12	98.1%	6.6%	11.4	2.9 (↓74.6%)	0.4	0.9
Biomodels	22,079	45.2%	66.4%	741.4	575.6 (↓22.4%)	540.1	2.6
cellCycle	1	$> 99.9\%$	$< 0.1\%$	–	13.9 (–)	<0.1	4.9
NCI+CHEBI	65	95.6%	10.3%	116.6	34.0 (↓70.8%)	16.3	4.1
NCI+GO	65	96.7%	10.4%	110.0	37.6 (↓65.8%)	17.6	3.2
NCI+Mouse	65	96.0%	13.3%	93.7	31.0 (↓66.9%)	16.6	2.6

OWL 2 EL — Data Retrieval Queries?

- PTime potentially problematical for very large datasets

OWL 2 EL — Data Retrieval Queries?

- PTime potentially problematical for very large datasets
- Various approaches:
  - Materialise taxonomy and use DBMS (incomplete reasoning)
  - "Combined approach" using materialisation + OBDA [2]
  - Datalog engine with (some form of) query rewriting [3]
  - Highly optimised ABox reasoners [4]

[2] Kontchakov, Lutz, Toman, Wolter, Zakharyaschev: The Combined Approach to Ontology-Based Data Access. IJCAI 2011.

[3] Stefanoni, Motik, Horrocks: Small Datalog Query Rewritings for EL. DL 2012

[4] Kazakov, Kroetzsch, Simancik: Practical Reasoning with Nominals in the EL Family of Description Logics. KR 2012

Various Approaches — Different Tradeoffs

❷ Use a suitable "profile" and specialised reasoner:

✓ LogSpace query answering (in size of data)

✓ Reliable answers (for inputs in the profile)

✗ Restricted expressivity of the ontology language

✗ Reasoners reject inputs outside profile

Various Approaches — Different Tradeoffs

❷ Use a suitable "profile" and specialised reasoner:

✓ LogSpace query answering (in size of data)

✓ Reliable answers (for inputs in the profile)

✗ Restricted expressivity of the ontology language

✗ Reasoners reject inputs outside profile

OWL 2 QL ontology reasoners:

- E.g., QuOnto, **Requiem**, ...
- Based on query rewriting technique — ontology used to rewrite (expand) query
- Targets applications where data stored in RDBMS — aka **Ontology Based Data Access** (OBDA)

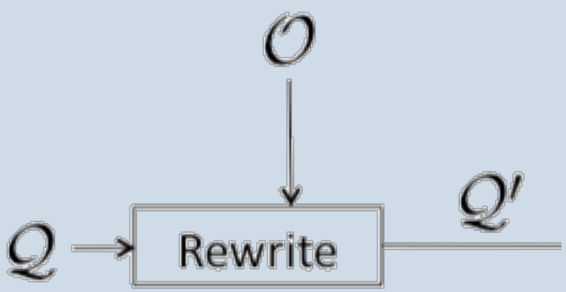
Query Rewriting — How Does It Work?

Given ontology $\mathcal{O}$ query $\mathcal{Q}$ and mappings $\mathcal{M}$:

Query Rewriting — How Does It Work?

Given ontology $\mathcal{O}$ query $\mathcal{Q}$ and mappings $\mathcal{M}$:

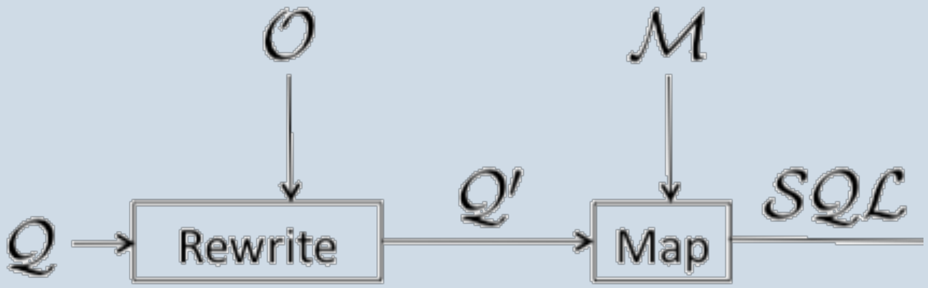
- **Rewrite** $\mathcal{Q} \rightarrow \mathcal{Q}'$ s.t. answering $\mathcal{Q}'$ without $\mathcal{O}$ equivalent to answering $\mathcal{Q}$ w.r.t. $\mathcal{O}$ *for any dataset*



Query Rewriting — How Does It Work?

Given ontology $\mathcal{O}$ query $\mathcal{Q}$ and mappings $\mathcal{M}$:

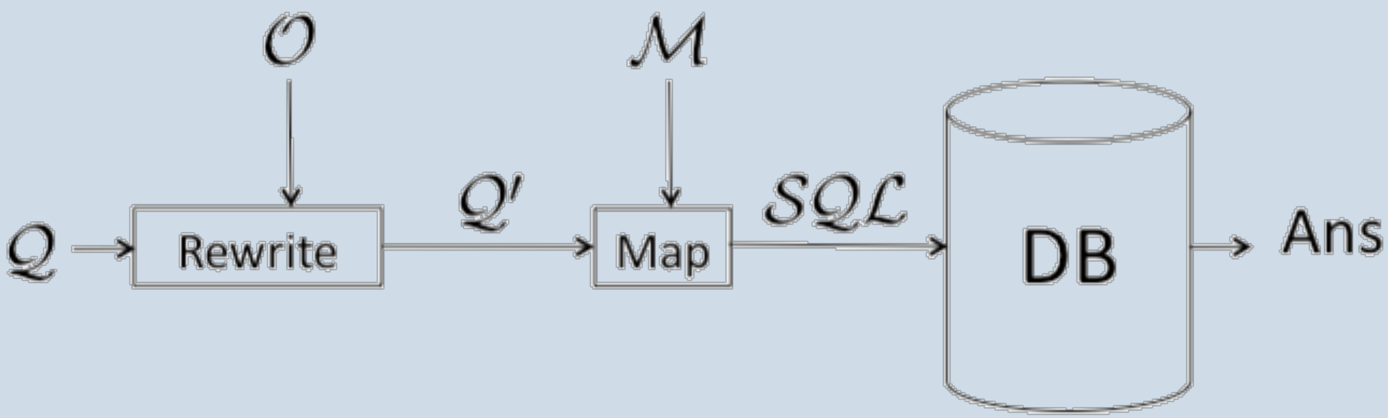
- **Rewrite** $\mathcal{Q} \rightarrow \mathcal{Q}'$ s.t. answering $\mathcal{Q}'$ without $\mathcal{O}$ equivalent to answering $\mathcal{Q}$ w.r.t. $\mathcal{O}$ *for any dataset*
- **Map** ontology queries → DB queries (typically SQL) using mappings $\mathcal{M}$ to rewrite $\mathcal{Q}$' into a DB query



Query Rewriting — How Does It Work?

Given ontology $\mathcal{O}$ query $\mathcal{Q}$ and mappings $\mathcal{M}$:

- **Rewrite** $\mathcal{Q} \rightarrow \mathcal{Q}'$ s.t. answering $\mathcal{Q}'$ without $\mathcal{O}$ equivalent to answering $\mathcal{Q}$ w.r.t. $\mathcal{O}$ *for any dataset*
- **Map** ontology queries → DB queries (typically SQL) using mappings $\mathcal{M}$ to rewrite $\mathcal{Q}$' into a DB query
- **Evaluate** (SQL) query against DB



Query Rewriting — Example

$$\mathcal{O} \begin{cases} \text{Doctor} \sqsubseteq \exists \text{treats}.\text{Patient} \\ \text{Consultant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\mathcal{Q} \quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$$

$$\mathcal{M} \begin{cases} \text{Doctor} & \mapsto & \text{SELECT Name FROM Doctor} \\ \text{Patient} & \mapsto & \text{SELECT Name FROM Patient} \\ \text{treats} & \mapsto & \text{SELECT DName, PName FROM Treats} \end{cases}$$

Query Rewriting — Example

$$\mathcal{O} \left\{ \begin{array}{r} \text{Doctor} \sqsubseteq \exists \text{treats.Patient} \\ \text{Consultant} \sqsubseteq \text{Doctor} \end{array} \right.$$

$$\mathcal{Q} \quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$$

$$Q' \left\{ \begin{array}{l} Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y) \\ Q(x) \leftarrow \text{Doctor}(x) \wedge \text{Patient}(f(x)) \\ Q(x) \leftarrow \text{treats}(x, f(x)) \wedge \text{Doctor}(x) \\ Q(x) \leftarrow \text{Doctor}(x) \\ Q(x) \leftarrow \text{Consultant}(x) \end{array} \right.$$

$$\mathcal{M} \left\{ \begin{array}{rcl} \text{Doctor} & \mapsto & \text{SELECT Name FROM Doctor} \\ \text{Patient} & \mapsto & \text{SELECT Name FROM Patient} \\ \text{treats} & \mapsto & \text{SELECT DName, PName FROM Treats} \end{array} \right.$$

Query Rewriting — Example

$$\mathcal{O}\begin{cases} \mathsf{Doctor} \sqsubseteq \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consultant} \sqsubseteq \mathsf{Doctor} \end{cases}$$

$$\mathcal{Q} \quad Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$$

Q':

- $Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$
- ~~$Q(x) \leftarrow \mathsf{Doctor}(x) \wedge \mathsf{Patient}(f(x))$~~
- ~~$Q(x) \leftarrow \mathsf{treats}(x, f(x)) \wedge \mathsf{Doctor}(x)$~~
- $Q(x) \leftarrow \mathsf{Doctor}(x)$
- $Q(x) \leftarrow \mathsf{Consultant}(x)$

$\mathcal{M}$:

Doctor	$\mapsto$	SELECT Name FROM Doctor
Patient	$\mapsto$	SELECT Name FROM Patient
treats	$\mapsto$	SELECT DName, PName FROM Treats

Query Rewriting — Example

$$\mathcal{O} \left\{ \begin{array}{r} \mathsf{Doctor} \sqsubseteq \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consultant} \sqsubseteq \mathsf{Doctor} \end{array} \right.$$

$$\mathcal{Q} \quad Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y)$$

$$Q' \left\{ \begin{array}{l} Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y) \\ \require{cancel}\cancel{Q(x) \leftarrow \mathsf{Doctor}(x) \wedge \mathsf{Patient}(f(x))} \\ \cancel{Q(x) \leftarrow \mathsf{treats}(x, f(x)) \wedge \mathsf{Doctor}(x)} \\ Q(x) \leftarrow \mathsf{Doctor}(x) \\ \cancel{Q(x) \leftarrow \mathsf{Consultant}(x)} \end{array} \right.$$

$$\mathcal{M} \left\{ \begin{array}{rcl} \mathsf{Doctor} & \mapsto & \text{SELECT Name FROM Doctor} \\ \mathsf{Patient} & \mapsto & \text{SELECT Name FROM Patient} \\ \mathsf{treats} & \mapsto & \text{SELECT DName, PName FROM Treats} \end{array} \right.$$

Query Rewriting — Example

$\mathcal{O}$:
- $\text{Doctor} \sqsubseteq \exists\text{treats}.\text{Patient}$
- $\text{Consultant} \sqsubseteq \text{Doctor}$

$\mathcal{Q}$: $Q(x) \leftarrow \text{treats}(x,y) \wedge \text{Patient}(y)$

Q':
- $Q(x) \leftarrow \text{treats}(x,y) \wedge \text{Patient}(y)$
- ~~$Q(x) \leftarrow \text{Doctor}(x) \wedge \text{Patient}(f(x))$~~
- ~~$Q(x) \leftarrow \text{treats}(x,f(x)) \wedge \text{Doctor}(x)$~~
- $Q(x) \leftarrow \text{Doctor}(x)$
- ~~$Q(x) \leftarrow \text{Consultant}(x)$~~

$\mathcal{M}$:
- Doctor $\mapsto$ SELECT Name FROM Doctor
- Patient $\mapsto$ SELECT Name FROM Patient
- treats $\mapsto$ SELECT DName, PName FROM Treats

$\mathcal{SQL}$:

SELECT Name FROM Doctor UNION
SELECT DName FROM Treats, Patient WHERE PName=Name

Query Rewriting — Issues

❶ Rewriting

- May be large (worst case exponential in size of ontology)
- Queries may be hard for existing DBMSs
- Ongoing work on OBDA optimisation techniques, e.g., [5]

❷ Mappings

- May be difficult to develop and maintain
- Little work in this area to date

[5] Rodriguez-Muro, Calvanese: High Performance Query Answering over DL-Lite Ontologies. KR 2012

Various Approaches — Different Tradeoffs

❸ Use full power of OWL and incomplete reasoner:

- ✓ Well-suited for modeling complex domains
- ✓ Favourable scalability properties
- ✓ Flexibility: no inputs rejected
- ✗ Incomplete answers (and degree of incompleteness not known)

OWL 2 RL ontology reasoners:

- E.g., Oracle's Semantic Datastore, Sesame, Jena, OWLim, ...
- Based on RDF triple stores and chase-like materialisation
- Widely used in practice to reason with large datasets
- Complete (only) for RL ontologies and ground atomic queries

Materialisation — How Does It Work?

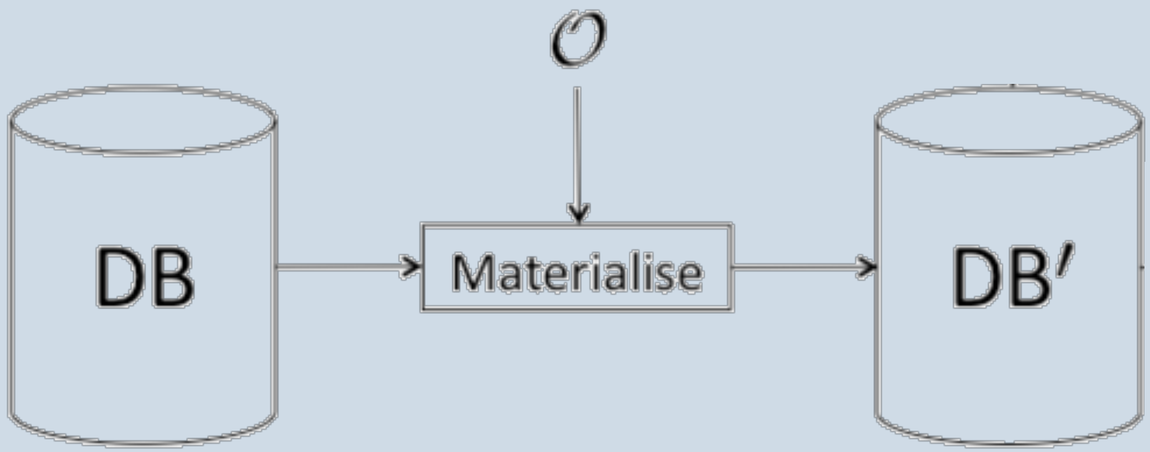
Given (RDF) data DB, ontology $\mathcal{O}$ and query $\mathcal{Q}$:

Materialisation — How Does It Work?

Given (RDF) data DB, ontology $\mathcal{O}$ and query $\mathcal{Q}$:

- **Materialise** (RDF) data DB $\rightarrow$ DB$'$ s.t. evaluating $\mathcal{Q}$ w.r.t. DB$'$ equivalent to answering $\mathcal{Q}$ w.r.t. DB and $\mathcal{O}$

  nb: Closely related to **chase** procedure used with DB dependencies



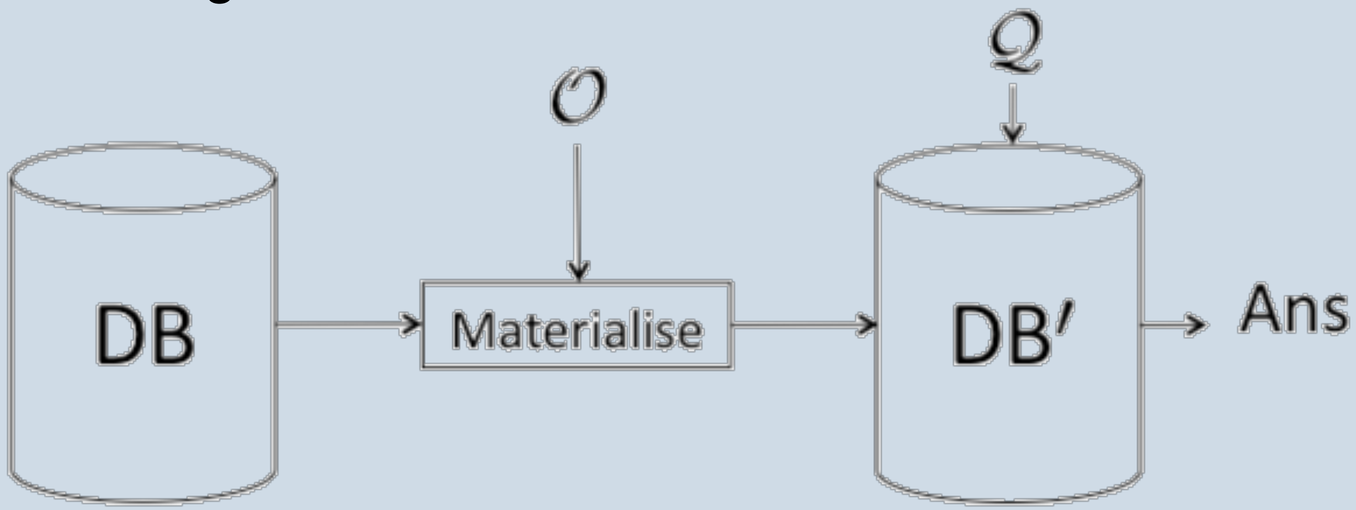
Materialisation — How Does It Work?

Given (RDF) data DB, ontology $\mathcal{O}$ and query $\mathcal{Q}$:

- **Materialise** (RDF) data DB $\rightarrow$ DB′ s.t. evaluating $\mathcal{Q}$ w.r.t. DB′ equivalent to answering $\mathcal{Q}$ w.r.t. DB and $\mathcal{O}$

  nb: Closely related to **chase** procedure used with DB dependencies

- **Evaluate** $\mathcal{Q}$ against DB′



Materialisation — Example

$$\mathcal{O} \begin{cases} \mathsf{Doctor} \equiv \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consulatant} \sqsubseteq \mathsf{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \end{cases}$$

Materialisation — Example

$$\mathcal{O} \begin{cases} \mathsf{Doctor} \equiv \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consulatant} \sqsubseteq \mathsf{Doctor} \end{cases}$$

$$\mathsf{DB} \begin{cases} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \end{cases}$$

$$\mathsf{DB}' \begin{cases} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \\ \mathsf{Doctor}(d_1) \\ \mathsf{Doctor}(c_1) \end{cases}$$

Materialisation — Example

$$\mathcal{O} \left\{ \begin{array}{r} \mathsf{Doctor} \equiv \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consulatant} \sqsubseteq \mathsf{Doctor} \end{array} \right.$$

$$\mathsf{DB} \left\{ \begin{array}{l} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \end{array} \right.$$

$$\mathsf{DB}' \left\{ \begin{array}{l} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \\ \mathsf{Doctor}(d_1) \\ \mathsf{Doctor}(c_1) \end{array} \right.$$

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \mathsf{Doctor}(y)$$

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Materialisation — Example

$$\mathcal{O} \left\{ \begin{array}{r} \text{Doctor} \equiv \exists \text{treats.Patient} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{array} \right.$$

$$\text{DB} \left\{ \begin{array}{l} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{array} \right. \qquad \text{DB}' \left\{ \begin{array}{l} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \end{array} \right.$$

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \qquad \rightsquigarrow \qquad \{d_2, d_1, c_1\}$$

Materialisation — Example

$$\mathcal{O} \begin{cases} \text{Doctor} \equiv \exists \text{treats}.\text{Patient} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases} \qquad \text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \end{cases}$$

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \qquad \rightsquigarrow \qquad \{d_2, d_1, c_1\}$$

$$\mathcal{Q}_2 \quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$$

Materialisation — Example

$$\mathcal{O}\left\{\begin{array}{r}\text{Doctor} \equiv \exists\text{treats}.\text{Patient}\\ \text{Consulatant} \sqsubseteq \text{Doctor}\end{array}\right.$$

$$\text{DB}\left\{\begin{array}{l}\text{treats}(d_1, p_1)\\ \text{Patient}(p_1)\\ \text{Doctor}(d_2)\\ \text{Consultant}(c_1)\end{array}\right.$$

$$\text{DB}'\left\{\begin{array}{l}\text{treats}(d_1, p_1)\\ \text{Patient}(p_1)\\ \text{Doctor}(d_2)\\ \text{Consultant}(c_1)\\ \text{Doctor}(d_1)\\ \text{Doctor}(c_1)\end{array}\right.$$

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \quad \rightsquigarrow \quad \{d_2, d_1, c_1\}$$

$$\mathcal{Q}_2 \quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y) \quad \rightsquigarrow \quad \{d_1\}$$

Dealing With Frequently Changing Data

Adding data is relatively easy

- Monotonicity of FOL means that extending existing materialisation is sound
- Can still be quite costly if naively implemented

Changing/retracting data is much harder

- Naive solution requires all materialised facts to be discarded
- Re-materialisation very costly for large data sets
- But incremental reasoning is possible using view maintenance based techniques [6]

[6] Motik, Horrocks, and Kim. Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In Proc. of WWW 2012.

Dealing with Incompleteness

- Materialisation based reasoning complete for **OWL 2 RL** profile (and ground atomic queries)
- But for ontologies **outside the profile**:
  - Reasoning may be incomplete
  - Incompleteness difficult to measure via empirical testing
- Possible solutions offered by recent work:
  - **Measuring and repairing incompleteness**
  - **Chase materialisation**
  - **Computing upper and lower bounds**

Measuring and Repairing Incompleteness

- Use ontology $\mathcal{O}$ (and query $\mathcal{Q}$) to generate a test suite

DEPARTMENT OF COMPUTER SCIENCE UNIVERSITY OF OXFORD
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Measuring and Repairing Incompleteness

- Use ontology $\mathcal{O}$ (and query $\mathcal{Q}$) to generate a test suite
- A **test suite** for $\mathcal{O}$ is a pair $\mathbf{S} = \langle \mathbf{S}_\bot, \mathbf{S}_Q \rangle$
  - $\mathbf{S}_\bot$ a set of ABoxes that are unsatisfiable w.r.t. $\mathcal{O}$
  - $\mathbf{S}_Q$ a set of paris $\langle \mathcal{A}, \mathcal{Y} \rangle$ with $\mathcal{A}$ an ABox and $\mathcal{Y}$ a query

Measuring and Repairing Incompleteness

- Use ontology $\mathcal{O}$ (and query $\mathcal{Q}$) to generate a test suite

- A **test suite** for $\mathcal{O}$ is a pair $\mathbf{S} = \langle \mathbf{S}_\bot, \mathbf{S}_Q \rangle$
  - $\mathbf{S}_\bot$ a set of ABoxes that are unsatisfiable w.r.t. $\mathcal{O}$
  - $\mathbf{S}_Q$ a set of paris $\langle \mathcal{A}, \mathcal{Y} \rangle$ with $\mathcal{A}$ an ABox and $\mathcal{Y}$ a query

- A **reasoner** $\mathcal{R}$ passes $\mathbf{S}$ if:
  - $\mathcal{R}$ finds $\mathcal{O} \cup \mathcal{A}$ unsatisfiable for each $\mathcal{A} \in \mathbf{S}_\bot$
  - $\mathcal{R}$ complete for $\mathcal{Y}$ w.r.t. $\mathcal{O} \cup \mathcal{A}$ for each $\langle \mathcal{A}, \mathcal{Y} \rangle \in \mathbf{S}_Q$

[7] Cuenca Grau, Motik, Stoilos, and Horrocks. Completeness Guarantees for Incomplete Ontology Reasoners: Theory and Practice. JAIR, 43:419-476, 2012.

Chase Materialisation

- Applicable to **acyclic** ontologies
  - Acyclicity can be checked using, e.g., graph based techniques (weak acyclicity, joint acyclicity, etc.)
  - Many realistic ontologies turn out to be acyclic
- Given acyclic ontology $\mathcal{O}$, can apply chase materialisation:
  - Ontology translated into existential rules (aka dependencies)
  - Existential rules can introduce fresh Skolem individuals
  - Termination guaranteed for acyclic ontologies

[8] Cuenca Grau et al. Acyclicity Conditions and their Application to Query Answering in Description Logics. In Proc. of KR 2012.

Chase Materialisation — Example

$$\mathcal{O} \left\{ \begin{array}{r} \text{Doctor} \equiv \exists\text{treats}.\text{Patient} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{array} \right.$$

$$\text{DB} \left\{ \begin{array}{l} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{array} \right.$$

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Chase Materialisation — Example

$$\mathcal{O} \begin{cases} \text{Doctor} \equiv \exists \text{treats}.\text{Patient} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_2, f(d_2)) \\ \text{Patient}(f(d_2)) \\ \text{treats}(c_1, f(c_1)) \\ \text{Patient}(f(c_1)) \end{cases}$$

Skolems

Chase Materialisation — Example

$$\mathcal{O} \begin{cases} \textsf{Doctor} \equiv \exists \textsf{treats}.\textsf{Patient} \\ \textsf{Consulatant} \sqsubseteq \textsf{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \textsf{treats}(d_1, p_1) \\ \textsf{Patient}(p_1) \\ \textsf{Doctor}(d_2) \\ \textsf{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \textsf{treats}(d_1, p_1) \\ \textsf{Patient}(p_1) \\ \textsf{Doctor}(d_2) \\ \textsf{Consultant}(c_1) \\ \textsf{Doctor}(d_1) \\ \textsf{Doctor}(c_1) \\ \textsf{treats}(d_2, f(d_2)) \\ \textsf{Patient}(f(d_2)) \\ \textsf{treats}(c_1, f(c_1)) \\ \textsf{Patient}(f(c_1)) \end{cases}$$

Skolems

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \textsf{Doctor}(y)$$

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Chase Materialisation — Example

$$\mathcal{O} \begin{cases} \text{Doctor} \equiv \exists \text{treats.Patient} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_2, f(d_2)) \\ \text{Patient}(f(d_2)) \\ \text{treats}(c_1, f(c_1)) \\ \text{Patient}(f(c_1)) \end{cases}$$

Skolems

$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \quad \rightsquigarrow \quad \{d_2, d_1, c_1\}$

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Chase Materialisation — Example

$$\mathcal{O} \begin{cases} \text{Doctor} \equiv \exists \text{treats.Patient} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_2, f(d_2)) \\ \text{Patient}(f(d_2)) \\ \text{treats}(c_1, f(c_1)) \\ \text{Patient}(f(c_1)) \end{cases}$$

Skolems (→ $\text{treats}(d_2, f(d_2))$, $\text{treats}(c_1, f(c_1))$)

$\mathcal{Q}_1$ $\quad Q(x) \leftarrow \text{Doctor}(y) \quad \rightsquigarrow \quad \{d_2, d_1, c_1\}$

$\mathcal{Q}_2$ $\quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$

Chase Materialisation — Example

$$\mathcal{O} \begin{cases} \mathsf{Doctor} \equiv \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consulatant} \sqsubseteq \mathsf{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \end{cases} \qquad \text{DB}' \begin{cases} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \\ \mathsf{Doctor}(d_1) \\ \mathsf{Doctor}(c_1) \\ \mathsf{treats}(d_2, f(d_2)) \\ \mathsf{Patient}(f(d_2)) \\ \mathsf{treats}(c_1, f(c_1)) \\ \mathsf{Patient}(f(c_1)) \end{cases}$$

Skolems

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \mathsf{Doctor}(y) \quad \rightsquigarrow \quad \{d_2, d_1, c_1\}$$

$$\mathcal{Q}_2 \quad Q(x) \leftarrow \mathsf{treats}(x, y) \wedge \mathsf{Patient}(y) \quad \rightsquigarrow \quad \{d_1, d_2, c_1\}$$

Computing Lower and Upper Bounds

- RL reasoning w.r.t. OWL ontology $\mathcal{O}$ gives lower bound answer ***L***

Computing Lower and Upper Bounds

- RL reasoning w.r.t. OWL ontology $\mathcal{O}$ gives lower bound answer ***L***
- Transform $\mathcal{O}$ into strictly stronger OWL RL ontology
  - Transform ontology into Datalog$^{\pm,\vee}$ rules
  - Eliminate $\vee$ by transforming to $\wedge$
  - Eliminate existentials by replacing with Skolem constants
  - Discard rules with empty heads
  - Transform rules into OWL 2 RL ontology $\mathcal{O}'$

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Computing Lower and Upper Bounds

- RL reasonting w.r.t. $\mathcal{O}$' gives (complete but unsound) upper bound answer ***U***

UNIVERSITY OF OXFORD DEPARTMENT OF COMPUTER SCIENCE
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
SEVENTH FRAMEWORK PROGRAMME

Computing Upper Bound — Example

$$\mathcal{O} \begin{cases} \mathsf{Doctor} \equiv \exists \mathsf{treats}.\mathsf{Patient} \\ \mathsf{Consulatant} \sqsubseteq \mathsf{Doctor} \end{cases}$$

$$\mathrm{DB} \begin{cases} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \end{cases}$$

Computing Upper Bound — Example

$$\mathcal{O}' \left\{ \begin{array}{r} \mathsf{Doctor} \sqsubseteq \exists \mathsf{treats}.\{P\} \\ \{\mathsf{P}\} \sqsubseteq \mathsf{Patient} \\ \exists \mathsf{treats}.\mathsf{Patient} \sqsubseteq \mathsf{Doctor} \\ \mathsf{Consulatant} \sqsubseteq \mathsf{Doctor} \end{array} \right.$$

$$\text{DB} \left\{ \begin{array}{l} \mathsf{treats}(d_1, p_1) \\ \mathsf{Patient}(p_1) \\ \mathsf{Doctor}(d_2) \\ \mathsf{Consultant}(c_1) \end{array} \right.$$

Computing Upper Bound — Example

$$\mathcal{O}' \left\{ \begin{array}{r} \text{Doctor} \sqsubseteq \exists \text{treats}.\{P\} \\ \{\text{P}\} \sqsubseteq \text{Patient} \\ \exists \text{treats}.\text{Patient} \sqsubseteq \text{Doctor} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{array} \right.$$

$$\text{DB} \left\{ \begin{array}{l} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{array} \right.$$

$$\text{DB}' \left\{ \begin{array}{l} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Patient}(P) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_1, P) \\ \text{treats}(d_2, P) \\ \text{treats}(c_1, P) \end{array} \right.$$

Computing Upper Bound — Example

$$\mathcal{O}' \begin{cases} \text{Doctor} \sqsubseteq \exists \text{treats}.\{P\} \\ \{\text{P}\} \sqsubseteq \text{Patient} \\ \exists \text{treats}.\text{Patient} \sqsubseteq \text{Doctor} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Patient}(P) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_1, P) \\ \text{treats}(d_2, P) \\ \text{treats}(c_1, P) \end{cases}$$

$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y)$

Computing Upper Bound — Example

$$\mathcal{O}' \begin{cases} \text{Doctor} \sqsubseteq \exists\text{treats}.\{P\} \\ \{\text{P}\} \sqsubseteq \text{Patient} \\ \exists\text{treats}.\text{Patient} \sqsubseteq \text{Doctor} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Patient}(P) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_1, P) \\ \text{treats}(d_2, P) \\ \text{treats}(c_1, P) \end{cases}$$

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \qquad \rightsquigarrow \qquad \{d_2, d_1, c_1\}$$

Computing Upper Bound — Example

$$\mathcal{O}' \begin{cases} \text{Doctor} \sqsubseteq \exists \text{treats}.\{P\} \\ \{\text{P}\} \sqsubseteq \text{Patient} \\ \exists \text{treats}.\text{Patient} \sqsubseteq \text{Doctor} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Patient}(P) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_1, P) \\ \text{treats}(d_2, P) \\ \text{treats}(c_1, P) \end{cases}$$

$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \qquad \rightsquigarrow \qquad \{d_2, d_1, c_1\}$

$\mathcal{Q}_2 \quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y)$

University of Oxford Department of Computer Science
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
Seventh Framework Programme

Computing Upper Bound — Example

$$\mathcal{O}' \begin{cases} \text{Doctor} \sqsubseteq \exists\text{treats}.\{P\} \\ \{P\} \sqsubseteq \text{Patient} \\ \exists\text{treats}.\text{Patient} \sqsubseteq \text{Doctor} \\ \text{Consulatant} \sqsubseteq \text{Doctor} \end{cases}$$

$$\text{DB} \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \end{cases}$$

$$\text{DB}' \begin{cases} \text{treats}(d_1, p_1) \\ \text{Patient}(p_1) \\ \text{Doctor}(d_2) \\ \text{Consultant}(c_1) \\ \text{Patient}(P) \\ \text{Doctor}(d_1) \\ \text{Doctor}(c_1) \\ \text{treats}(d_1, P) \\ \text{treats}(d_2, P) \\ \text{treats}(c_1, P) \end{cases}$$

$$\mathcal{Q}_1 \quad Q(x) \leftarrow \text{Doctor}(y) \quad \rightsquigarrow \quad \{d_2, d_1, c_1\}$$

$$\mathcal{Q}_2 \quad Q(x) \leftarrow \text{treats}(x, y) \wedge \text{Patient}(y) \quad \rightsquigarrow \quad \{d_1, d_2, c_1\}$$

Computing Lower and Upper Bounds

- RL reasonting w.r.t. $\mathcal{O}'$ gives (complete but unsound) upper bound answer ***U***

- If ***L*** = ***U***, then both answers are sound and complete
- If ***L*** ≠ ***U***, then ***U*** \ ***L*** identifies a (small) set of "possible" answers
  - Indicates range of uncertainty
  - Can (more efficiently) check possible answers using, e.g., HermiT
  - Future work: use ***U*** \ ***L*** to identify (small) "relevant" subset of data needed to efficiently compute exact answer

[9] Zhou, Cuenca Grau, and Horrocks. Efficient Upper Bound Computation of Query Answers in Expressive Description Logics. In Proc. of DL 2012, volume 846 of CEUR.

Discussion

Numerous **exciting developments** & research areas

- Rewriting: optimisations, extensions (datalog engines), etc.
- Materialisation: chase, repair, truth maintenance, upper bounds etc.
- Combined techniques (materialisation+rewriting), Datalog
- Specialised RDF stores, Column stores, massive parallelism, etc.
- Parameterised complexity, new query evaluation techniques, etc.

Discussion

Numerous **exciting developments** & research areas

- Rewriting: optimisations, extensions (datalog engines), etc.
- Materialisation: chase, repair, truth maintenance, upper bounds etc.
- Combined techniques (materialisation+rewriting), Datalog
- Specialised RDF stores, Column stores, massive parallelism, etc.
- Parameterised complexity, new query evaluation techniques, etc.

Consider **progress on schema reasoning**:

Year	$\mathcal{O}$-size	Complete	Time (s)
1995	3,000	No	10^5
1998	3,000	Yes	300
2005	30,000	Yes	30
2010	400,000	Yes	5

Discussion

Numerous **exciting developments** & research areas

- Rewriting: optimisations, extensions (datalog engines), etc.
- Materialisation: chase, repair, truth maintenance, upper bounds etc.
- Combined techniques (materialisation+rewriting), Datalog
- Specialised RDF stores, Column stores, massive parallelism, etc.
- Parameterised complexity, new query evaluation techniques, etc.

Consider **progress on schema reasoning**:

Year	Concepts	Complete	Time (s)
1995	3,000	No	10^5
1998	[illegible]	[illegible]	[illegible]
2005	30,000	Yes	30
2010	400,000	Yes	5

Looking forward to similar progress on query answering!

Discussion

Numerous **exciting developments** & research areas

- Rewriting: optimisations, extensions (datalog engines), etc.
- Materialisation: chase, repair, truth maintenance, upper bounds etc.
- Hybrid techniques (materialisation+rewriting), Datalog
- Specialised RDF stores, Column stores, massive parallelism, etc.
- Parameterised complexity, new query evaluation techniques, etc.

$$\text{Semantics} \sqcap \text{Scalability} \not\models \bot \ !$$

Consider **progress on schema reasoning**:

Year	$\mathcal{O}$-size	Complete	Time (s)
1995	3,000	No	10^5
1998	3,000	Yes	300
2005	30,000	Yes	30
2010	400,000	Yes	5

Acknowledgements

EPSRC Engineering and Physical Sciences Research Council

SIXTH FRAMEWORK PROGRAMME

SEVENTH FRAMEWORK PROGRAMME

DARPA

References

[1] Armas Romero, Cuenca Grau, and Horrocks. Modular Combination of Reasoners for Ontology Classification. In Proc. of ISWC 2012 (to appear).

[2] Kontchakov, Lutz, Toman, Wolter, Zakharyaschev: The Combined Approach to Ontology-Based Data Access. IJCAI 2011.

[3] Stefanoni, Motik, Horrocks: Small Datalog Query Rewritings for EL. DL 2012

[4] Kazakov, Kroetzsch, Simancik: Practical Reasoning with Nominals in the EL Family of Description Logics. KR 2012

[5] Rodriguez-Muro, Calvanese: High Performance Query Answering over DL-Lite Ontologies. KR 2012

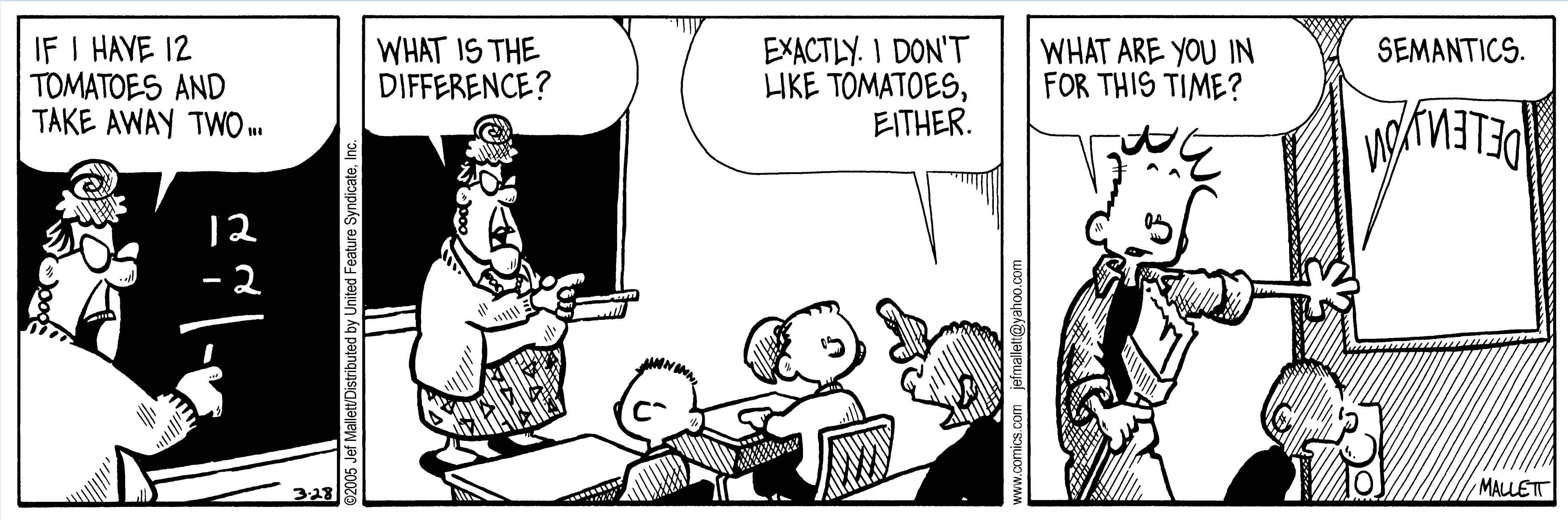
[6] Motik, Horrocks, and Kim. Delta-reasoner: a semantic web reasoner for an intelligent mobile platform. In Proc. of WWW 2012.

[7] Cuenca Grau, Motik, Stoilos, and Horrocks. Completeness Guarantees for Incomplete Ontology Reasoners: Theory and Practice. JAIR, 43:419-476, 2012

[8] Cuenca Grau et al. Acyclicity Conditions and their Application to Query Answering in Description Logics. In Proc. of KR 2012.

[9] Zhou, Cuenca Grau, and Horrocks. Efficient Upper Bound Computation of Query Answers in Expressive Description Logics. In Proc. of DL 2012

Thank you for listening



Any questions?

University of Oxford Department of Computer Science
Information Systems Group
EPSRC Engineering and Physical Sciences Research Council
Seventh Framework Programme