Semantics \setminus Scalability \models \bot ?

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"
Semantic Technologies

- Initial focus was on necessary underpinning, including:
Semantic Technologies

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

![Diagram showing Semantic Technologies logos](image-url)
Semantic Technologies

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

![Hermit](hermit.png) ![FaCT++](fact.png) ![ORACLE](oracle.png)

![Racer](racer.png) ![pellet](pellet.png) ![uOnto](uonto.png)

![OWLIM](owlim.png) ![CEL](cel.png) ![Jena](jena.png)
Semantic Technologies

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

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

- Resulting **robust infrastructure** used in SW applications
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?

1 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
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?

1. http://www.w3.org/2000/10/swappim/contact#Person
2. http://www.w3.org/1999/02/22-rdf-syntax-ns#type
3. http://www.w3.org/2000/10/swappim/contact#fullName
4. http://www.w3.org/2000/10/swappim/contact#mailbox
5. mailto:em@w3.org
6. Dr.
7. http://www.w3.org/2000/10/swappim/contact#personalTitle
How Does it Work?

2. Standardised language for exchanging vocabularies/schemas
   - W3C standard for vocabulary/schema exchange is OWL
   - OWL provides for rich conceptual schemas, aka ONTOLOGIES

\[
\text{Heart} \sqsubseteq \text{MuscularOrgan} \sqsupseteq \\
\exists \text{isPartOf}.\text{CirculatorySystem} \\
\text{HeartDisease} \equiv \text{Disease} \sqsupseteq \\
\exists \text{affects}.\text{Heart} \\
\text{VascularDisease} \equiv \text{Disease} \sqsupseteq \\
\exists \text{affects}.(\exists \text{isPartOf}.\text{CirculatorySystem})
\]
How Does it Work?

\[ \langle \text{John rdf:type Patient} \rangle \]
\[ \langle \text{John suffersFrom d1} \rangle \]
\[ \langle \text{d1 rdf:type HeartDisease} \rangle \]
How Does it Work?

Patients suffering from vascular disease

\[
\text{John rdf:type Patient} \\
\text{John suffersFrom d1} \\
\text{d1 rdf:type HeartDisease}
\]
How Does it Work?

Patients suffering from vascular disease

\{John rdf:type Patient\}
\{John suffersFrom d1\}
\{d1 rdf:type HeartDisease\}
How Does it Work?

Patients suffering from vascular disease

\[\text{John rdfs:typ Patient}\]
\[\text{John suffersFrom d1}\]
\[\text{d1 rdfs:typ HeartDisease}\]
How Does it Work?

Patients suffering from vascular disease

```
(John rdf:type Patient)
(John suffersFrom d1)
(d1 rdf:type HeartDisease)

Heart ∈ MuscularOrgan □
   □ isPartOf.CirculatorySystem
HeartDisease ≡ Disease □
   □ affects.Heart
VascularDisease ≡ Disease □
   □ affects.(□ isPartOf.CirculatorySystem)
```
How Does it Work?

Patients suffering from vascular disease

\[
\begin{align*}
\text{John} & \text{ rdf:type Patient} \\
\text{John} & \text{ suffersFrom d1} \\
\text{d1} & \text{ rdf:type HeartDisease}
\end{align*}
\]

\[
\begin{align*}
\text{Heart} & \text{ MuscularOrgan} \\
& \text{ isPartOf CirculatorySystem} \\
\text{HeartDisease} & \text{ Disease} \\
& \text{ affects Heart} \\
\text{VascularDisease} & \text{ Disease} \\
& \text{ affects (isPartOf CirculatorySystem)}
\end{align*}
\]
How Does it Work?

Patients suffering from vascular disease

John

\[
\langle \text{John rdf:type Patient} \rangle
\]

\[
\langle \text{John suffersFrom d1} \rangle
\]

\[
\langle \text{d1 rdf:type HeartDisease} \rangle
\]

Heart ⊑ MuscularOrgan ⊑
  \langle \text{isPartOf CirculatorySystem} \rangle

HeartDisease ⊑ Disease ⊑
  \langle \text{affects Heart} \rangle

VascularDisease ⊑ Disease ⊑
  \langle \text{affects } \langle \text{isPartOf CirculatorySystem} \rangle \rangle

How Does it Work?

Is heart disease a kind of vascular disease?

\[
\text{John rdf:type Patient} \\
\text{John suffersFrom d1} \\
\text{d1 rdf:type HeartDisease}
\]

\[
\text{Heart } \sqsubset \text{MuscularOrgan} \\
\quad \sqsubset \text{IsPartOf.CirculatorySystem} \\
\text{HeartDisease } \equiv \text{Disease} \\
\quad \sqsubset \text{Affects.Heart} \\
\text{VascularDisease } \equiv \text{Disease} \\
\quad \sqsubset \text{Affects.(IsPartOf.CirculatorySystem)}
\]
How Does it Work?

Is heart disease a kind of vascular disease?

\[ \text{John rdf:type Patient} \]
\[ \text{John suffersFrom d1} \]
\[ \text{d1 rdf:type HeartDisease} \]

- Heart \sqsubset \text{MuscularOrgan} \sqsubset \text{IsPartOf.CirculatorySystem}
- HeartDisease \sqsubset \text{Disease} \sqsubset \text{Affects.Heart}
- VascularDisease \sqsubset \text{Disease} \sqsubset \text{Affects.\{(IsPartOf.CirculatorySystem)\}}

YES
How Does it Work?

\( \{ \text{John } \text{rdf:type } \text{Patient} \} \)
\( \{ \text{John } \text{suffersFrom } \text{d1} \} \)
\( \{ \text{d1 } \text{rdf:type } \text{HeartDisease} \} \)

\text{Heart} \equiv \text{MuscularOrgan} \equiv \text{isPartOf.CirculatorySystem}
\text{HeartDisease} \equiv \text{Disease} \equiv \text{affects.Heart}
\text{VascularDisease} \equiv \text{Disease} \equiv \text{affects.(isPartOf.CirculatorySystem)}

Why?
How Does it Work?

\[
\text{\{John rdf:type Patient\}} \\
\text{\{John suffersFrom d1\}} \\
\text{\{d1 rdf:type HeartDisease\}}
\]

- \text{Heart} \sqsubseteq \text{MuscularOrgan} \sqsubseteq \text{IsPartOf.CirculatorySystem}
- \text{HeartDisease} \sqsubseteq \text{Disease} \sqsubseteq \text{Affects.Hart}
- \text{VascularDisease} \sqsubseteq \text{Disease} \sqsubseteq \text{Affects.(IsPartOf.CirculatorySystem)}

Why?
Applications: Semantic Web
Applications: Semantic Web
Applications: Semantic Web
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 Mobile 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 ⟷ Practice
Theory ~⇒ Practice

- OWL based on description logic SROIQ
Theory ↔ Practice

- OWL based on description logic 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 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-complete combined complexity
  - NP-hard data complexity (-v- logspace for databases)
Theory ⟷ Practice

- OWL based on description logic 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-complete combined complexity
  - NP-hard data complexity (-v- logspace for databases)

How can we provide robustly scalable query answering?
Various Approaches — Different Tradeoffs

1. 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
Various Approaches — Different Tradeoffs

2 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

2. 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
### Schema Reasoning — Solved Problem?

<table>
<thead>
<tr>
<th></th>
<th>SNOMED CT</th>
<th>GALLEN</th>
<th>FMA</th>
<th>GO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>$EL$</td>
<td>$EL$</td>
<td>$EL$</td>
<td>$EL$</td>
</tr>
<tr>
<td>#classes</td>
<td>315,489</td>
<td>23,136</td>
<td>78,977</td>
<td>19,468</td>
</tr>
<tr>
<td>#properties</td>
<td>58</td>
<td>950</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>#axioms</td>
<td>430,844</td>
<td>36,547</td>
<td>121,712</td>
<td>28,897</td>
</tr>
<tr>
<td>#⊆</td>
<td>$&gt;10^{11}$</td>
<td>$&gt;10^8$</td>
<td>$&gt;10^9$</td>
<td>$&gt;10^8$</td>
</tr>
<tr>
<td>ELK (1 worker)</td>
<td>13.15</td>
<td>1.33</td>
<td>0.44</td>
<td>0.20</td>
</tr>
<tr>
<td>ELK (4 workers)</td>
<td>5.02</td>
<td>0.77</td>
<td>0.39</td>
<td>0.19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Plant Anat.</th>
<th>SWEET-P</th>
<th>NCI-2</th>
<th>DOLCE-P</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>$SHIF$</td>
<td>$SHOIN$</td>
<td>$ALCH$</td>
<td>$SHOIN$</td>
</tr>
<tr>
<td>#classes</td>
<td>19,145</td>
<td>1,728</td>
<td>70,576</td>
<td>118</td>
</tr>
<tr>
<td>#properties</td>
<td>82</td>
<td>145</td>
<td>189</td>
<td>264</td>
</tr>
<tr>
<td>#axioms</td>
<td>35,770</td>
<td>2,419</td>
<td>100,304</td>
<td>265</td>
</tr>
<tr>
<td>#⊆</td>
<td>$&gt;10^8$</td>
<td>$&gt;10^6$</td>
<td>$&gt;10^9$</td>
<td>$&gt;10^4$</td>
</tr>
<tr>
<td>HermiT</td>
<td>11.2</td>
<td>11.2</td>
<td>—</td>
<td>105.1</td>
</tr>
<tr>
<td>Pellet</td>
<td>87.2</td>
<td>—</td>
<td>172.0</td>
<td>105.1</td>
</tr>
<tr>
<td>FaCT++</td>
<td>22.9</td>
<td>0.2</td>
<td>60.7</td>
<td>—</td>
</tr>
</tbody>
</table>
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
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]

MORe Modular Reasoner

| Ontology       | $|\mathcal{O} \setminus \mathcal{O}_L|$ | $|\Sigma^L|$ | $|\mathcal{M}_{|\mathcal{O},\Sigma^L|}|$ | Classif. time (seconds) | MORe    |
|----------------|----------------------------------|--------------|---------------------------------|------------------------|---------|
|                |                                  |              |                                 | HeraiT                | total   |
| GO             | 0                                | 100%         | 0%                              | 7.1                    | 2.2 (\textit{69.0\%}) |
| Gazetteer      | 0                                | 100%         | 0%                              | 838.1                  | 28.2 (\textit{96.6\%}) |
| NCI Protein    | 65                               | 94.9%        | 15.4%                           | 84.1                   | 28.6 (\textit{66.0\%}) |
| Biomodels      | 22,079                           | 45.2%        | 66.4%                           | 741.4                  | 575.6 (\textit{122.4\%}) |
| cellCycle      | 1                                | > 99.9%      | < 0.1%                          | 13.9 \textit{(--)}     | <0.1 |
| NCI+CHEBI      | 65                               | 95.6%        | 10.3%                           | 116.6                  | 34.0 (\textit{70.8\%}) |
| NCI+GO         | 65                               | 96.7%        | 10.4%                           | 110.0                  | 37.6 (\textit{65.8\%}) |
| NCI+Mouse      | 65                               | 96.0%        | 13.3%                           | 93.7                   | 31.0 (\textit{66.9\%}) |
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]


Various Approaches — Different Tradeoffs

2 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

② 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 $Q$ and mappings $M$: 
Query Rewriting — How Does It Work?

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

- **Rewrite** $Q \rightarrow Q'$ s.t. answering $Q'$ without $\mathcal{O}$ equivalent to answering $Q$ w.r.t. $\mathcal{O}$ for any dataset
Query Rewriting — How Does It Work?

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

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

- **Map** ontology queries \( \rightarrow \) DB queries (typically SQL) using mappings \( M \) to rewrite \( Q' \) into a DB query
Query Rewriting — How Does It Work?

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

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

![Diagram showing the process of query rewriting and evaluation](image)
Query Rewriting — Example

\[ O \{ \]
- Doctor \sqsubseteq \exists \text{treats.Patient} \\
- Consultant \sqsubseteq \text{Doctor} \\
\[ Q \{ Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \}

\[ M \{ \]
- Doctor \mapsto \text{SELECT Name FROM Doctor} \\
- Patient \mapsto \text{SELECT Name FROM Patient} \\
- treats \mapsto \text{SELECT DName, PName FROM Treats} \]
Query Rewriting — Example

\[
O \begin{cases}
  \text{Doctor} \subseteq \exists \text{treats}. \text{Patient} \\
  \text{Consultant} \subseteq \text{Doctor}
\end{cases}
\]

\[
Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
\]

\[
Q'(x) \leftarrow \\
\begin{cases}
  \text{treats}(x, y) \land \text{Patient}(y) \\
  \text{Doctor}(x) \land \text{Patient}(f(x)) \\
  \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
  \text{Doctor}(x) \\
  \text{Consultant}(x)
\end{cases}
\]

\[
M \begin{cases}
  \text{Doctor} \mapsto \text{SELECT Name FROM Doctor} \\
  \text{Patient} \mapsto \text{SELECT Name FROM Patient} \\
  \text{treats} \mapsto \text{SELECT DName, PName FROM Treats}
\end{cases}
\]
Query Rewriting — Example

\[ Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]

\[ Q'(x) \leftarrow \begin{align*}
    & \text{treats}(x, y) \land \text{Patient}(y) \\
    & \text{Doctor}(x) \land \text{Patient}(f(x)) \\
    & \text{treats}(x, f(x)) \land \text{Doctor}(x) \\
    & \text{Doctor}(x) \\
    & \text{Consultant}(x)
\end{align*} \]

\[ M \left\{ \begin{array}{l}
    \text{Doctor} \mapsto \text{SELECT Name FROM Doctor} \\
    \text{Patient} \mapsto \text{SELECT Name FROM Patient} \\
    \text{treats} \mapsto \text{SELECT DName, PName FROM Treats}
\end{array} \]
Query Rewriting — Example

\[ O \{ \quad \text{Doctor} \sqsubseteq \exists \text{treats. Patient} \]
\[ \quad \text{Consultant} \sqsubseteq \text{Doctor} \]
\[ Q \quad Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]

\[ Q' \{ \quad Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]
\[ \quad Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \]
\[ \quad Q(x) \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \]
\[ \quad Q(x) \leftarrow \text{Doctor}(x) \]
\[ \quad Q(x) \leftarrow \text{Consultant}(x) \]

\[ M \{ \quad \text{Doctor} \rightarrow \text{SELECT Name FROM Doctor} \]
\[ \quad \text{Patient} \rightarrow \text{SELECT Name FROM Patient} \]
\[ \quad \text{treats} \rightarrow \text{SELECT DName, PName FROM Treats} \]
Query Rewriting — Example

\[ Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]

\[ Q(x) \leftarrow \text{Doctor}(x) \land \text{Patient}(f(x)) \]

\[ Q(x) \leftarrow \text{treats}(x, f(x)) \land \text{Doctor}(x) \]

\[ Q(x) \leftarrow \text{Doctor}(x) \]

\[ Q(x) \leftarrow \text{Consultant}(x) \]

\[ M \]

- Doctor \rightarrow \text{SELECT Name FROM Doctor}
- Patient \rightarrow \text{SELECT Name FROM Patient}
- treats \rightarrow \text{SELECT DName, PName FROM Treats}

\[ \text{SELECT Name FROM Doctor UNION SELECT DName FROM Treats, Patient WHERE PName=Name} \]
Query Rewriting — Issues

1. **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]

2. **Mappings**
   - May be difficult to develop and maintain
   - Little work in this area to date

Various Approaches — Different Tradeoffs

3 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 $O$ and query $Q$:

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

  nb: Closely related to chase procedure used with DB dependencies
Materialisation — How Does It Work?

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

- **Materialise** (RDF) data DB $\rightarrow$ DB’ s.t. evaluating $Q$ w.r.t. DB’ equivalent to answering $Q$ w.r.t. DB and $\mathcal{O}$
  
  nb: Closely related to chase procedure used with DB dependencies

- **Evaluate** $Q$ against DB’
Materialisation — Example

\( O \) \{ 
\hspace{1cm} \text{Doctor} \equiv \exists \text{treats.Patient} \\
\hspace{1cm} \text{Consultant} \sqsubseteq \text{Doctor} \\
\}

\( DB \) \{ 
\hspace{1cm} \text{treats}\(d_1, p_1\) \\
\hspace{1cm} \text{Patient}(p_1) \\
\hspace{1cm} \text{Doctor}(d_2) \\
\hspace{1cm} \text{Consultant}(c_1) \\
\}
Materialisation — Example

\[ O \{ \]
\[ \text{Doctor} \equiv \exists \text{treats}. \text{Patient} \]
\[ \text{Consultant} \sqsubseteq \text{Doctor} \]
\[ \{ \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Consultant}(c_1) \]
\[ \} \]
\[ \}

\[ DB \{ \]
\[ \{ \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Consultant}(c_1) \]
\[ \} \]
\[ \}

\[ DB' \{ \]
\[ \{ \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Consultant}(c_1) \]
\[ \} \]
\[ \}

Materialisation — Example

\[ O \{ \]

\[ \text{Doctor} \equiv \exists \text{treats. Patient} \]

\[ \text{Consultant} \sqsubseteq \text{Doctor} \]

\[ \text{treats}(d_1, p_1) \]

\[ \text{Patient}(p_1) \]

\[ \text{Doctor}(d_2) \]

\[ \text{Consultant}(c_1) \]

\[ Q_1 \quad Q(x) \leftarrow \text{Doctor}(y) \]

\[ \{ \]

\[ \text{treats}(d_1, p_1) \]

\[ \text{Patient}(p_1) \]

\[ \text{Doctor}(d_2) \]

\[ \text{Consultant}(c_1) \]

\[ \text{Doctor}(d_1) \]

\[ \text{Doctor}(c_1) \]
Materialisation — Example

\[ O \{ \exists \text{treats. Patient} \]
\[
\text{Consultant} \subseteq \text{Doctor}
\]

\[ DB \{ \text{treats}(d_1, p_1), \quad \text{Patient}(p_1), \quad \text{Doctor}(d_2), \quad \text{Consultant}(c_1) \}
\]

\[ Q_1 \quad Q(x) \leftarrow \text{Doctor}(y) \]

\[ DB' \{ \text{treats}(d_1, p_1), \quad \text{Patient}(p_1), \quad \text{Doctor}(d_2), \quad \text{Consultant}(c_1), \quad \text{Doctor}(d_1), \quad \text{Doctor}(c_1) \}
\]

\[ \leadsto \quad \{d_2, d_1, c_1\} \]
Materialisation — Example

\( O \) \{ 
  \text{Doctor} \equiv \exists \text{treats}. \text{Patient} \\
  \text{Consultant} \subseteq \text{Doctor} \\
  \text{treats}(d_1, p_1) \\
  \text{Patient}(p_1) \\
  \text{Doctor}(d_2) \\
  \text{Consultant}(c_1) 
\}

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

\( Q_1 \) \( Q(x) \leftarrow \text{Doctor}(y) \)

\( Q_2 \) \( Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \)

\( \text{DB}' \) \{ 
  \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) \\
  \implies \{ d_2, d_1, c_1 \} 
\}
Materialisation — Example

\( \emptyset \) \[ \text{Doctor} \equiv \exists \text{treats}. \text{Patient} \]
\( \text{Consultant} \subseteq \text{Doctor} \)

\( \text{DB} \)
- \( \text{treats}(d_1, p_1) \)
- \( \text{Patient}(p_1) \)
- \( \text{Doctor}(d_2) \)
- \( \text{Consultant}(c_1) \)

\( \text{DB}' \)
- \( \text{treats}(d_1, p_1) \)
- \( \text{Patient}(p_1) \)
- \( \text{Doctor}(d_2) \)
- \( \text{Consultant}(c_1) \)
- \( \text{Doctor}(d_1) \)
- \( \text{Doctor}(c_1) \)

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

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

\( \leadsto \{d_2, d_1, c_1\} \)

\( \leadsto \{d_1\} \)
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]

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 $Q$) to generate a test suite
Measuring and Repairing Incompleteness

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

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

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

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

Chase Materialisation — Example

$\emptyset\{\text{Doctor} \equiv \exists \text{treats} . \text{Patient} \}
\text{Consultant} \subseteq \text{Doctor}

\text{DB}\{\text{treats}(d_1, p_1), \text{Patient}(p_1), \text{Doctor}(d_2), \text{Consultant}(c_1)\}
Chase Materialisation — Example

\[\emptyset \{\]
\[
\text{Doctor} \equiv \exists \text{treats}. \text{Patient}
\]
\[
\text{Consultant} \sqsubseteq \text{Doctor}
\]
\[
\text{treats}(d_1, p_1)
\]
\[
\text{Patient}(p_1)
\]
\[
\text{Doctor}(d_2)
\]
\[
\text{Consultant}(c_1)
\]

\[\text{DB} \{\]
\[
\text{treats}(d_1, p_1)
\]
\[
\text{Patient}(p_1)
\]
\[
\text{Doctor}(d_2)
\]
\[
\text{Consultant}(c_1)
\]

\[\text{DB}' \{\]
\[
\text{treats}(d_1, p_1)
\]
\[
\text{Patient}(p_1)
\]
\[
\text{Doctor}(d_2)
\]
\[
\text{Consultant}(c_1)
\]
\[
\text{treats}(d_2, f(d_2))
\]
\[
\text{Patient}(f(d_2))
\]
\[
\text{treats}(c_1, f(c_1))
\]
\[
\text{Patient}(f(c_1))
\]

\[\text{Skolems}\]
Chase Materialisation — Example

\[ O \{ \]

- Doctor \equiv \exists \text{treats} \cdot \text{Patient}
- Consultant \subseteq \text{Doctor}

\[ DB \{ \]
- \text{treats}(d_1, p_1)
- \text{Patient}(p_1)
- \text{Doctor}(d_2)
- \text{Consultant}(c_1)

\[ DB' \{ \]
- \text{treats}(d_1, p_1)
- \text{Patient}(p_1)
- \text{Doctor}(d_2)
- \text{Consultant}(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))

\[ Q_1 \quad Q(x) \leftarrow \text{Doctor}(y) \]

\[ \rightarrow \text{Skolems} \]
Chase Materialisation — Example

\[ Q_1 \quad Q(x) \leftarrow \text{Doctor}(y) \]

\[ \emptyset \{
\text{Doctor} \equiv \exists \text{treats}. \text{Patient}
\}
\]
\[ \text{Consultant} \subseteq \text{Doctor} \]

\[ \text{DB} \{ \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Consultant}(c_1) \]
\[ \} \]

\[ \text{DB'} \{ \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Doctor}(d_1) \]
\[ \text{Consultant}(c_1) \]
\[ \text{treats}(d_2, f(d_2)) \]
\[ \text{Patient}(f(d_2)) \]
\[ \text{treats}(c_1, f(c_1)) \]
\[ \text{Patient}(f(c_1)) \]
\[ \} \]

\[ \leadsto \{ d_2, d_1, c_1 \} \]

Skolems
Chase Materialisation — Example

O
\{ 
  \text{Doctor} \equiv \exists \text{treats. Patient} \\
  \text{Consultant} \subseteq \text{Doctor} \\
\}

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

DB' \{ 
  \text{treats}(d_1, p_1) \\
  \text{Patient}(p_1) \\
  \text{Doctor}(d_2) \\
  \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)) \\
\} \\
\sim \{ d_2, d_1, c_1 \}

Q_1 \quad Q(x) \leftarrow \text{Doctor}(y) \\
Q_2 \quad Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y)
Chase Materialisation — Example

\[ O \{ \]
\[ \text{Doctor} \equiv \exists \text{treats. Patient} \]
\[ \text{Consultant} \subseteq \text{Doctor} \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Consultant}(c_1) \]
\[ DB \{ \]
\[ Q_1 : Q(x) \leftarrow \text{Doctor}(y) \]
\[ Q_2 : Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \]
\[ DB' \{ \]
\[ \text{treats}(d_1, p_1) \]
\[ \text{Patient}(p_1) \]
\[ \text{Doctor}(d_2) \]
\[ \text{Consultant}(c_1) \]
\[ \text{treats}(d_2, f(d_2)) \]
\[ \text{Patient}(f(d_2)) \]
\[ \text{treats}(c_1, f(c_1)) \]
\[ \text{Patient}(f(c_1)) \]
\[ \leadsto \{ d_2, d_1, c_1 \} \]
\[ \leadsto \{ d_1, d_2, c_1 \} \]
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$^{\pm,v}$ rules
  - Eliminate $\lor$ by transforming to $\land$
  - Eliminate existentials by replacing with Skolem constants
  - Discard rules with empty heads
  - Transform rules into OWL 2 RL ontology $\mathcal{O'}$
Computing Lower and Upper Bounds

- RL reasoning w.r.t. $\mathcal{O}'$ gives (complete but unsound) upper bound answer $U$
Computing Lower and Upper Bounds

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

- If $L = U$, then both answers are sound and complete

- If $L \neq U$, then $U \setminus L$ identifies a (small) set of “possible” answers
  - Indicates range of uncertainty
  - Can (more efficiently) check possible answers using, e.g., HermiT
  - Future work: use $U \setminus L$ to identify (small) “relevant” subset of data needed to efficiently compute exact answer

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**:

<table>
<thead>
<tr>
<th>Year</th>
<th>$O$-size</th>
<th>Complete</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>3,000</td>
<td>No</td>
<td>$10^5$</td>
</tr>
<tr>
<td>1998</td>
<td>3,000</td>
<td>Yes</td>
<td>300</td>
</tr>
<tr>
<td>2005</td>
<td>30,000</td>
<td>Yes</td>
<td>30</td>
</tr>
<tr>
<td>2010</td>
<td>400,000</td>
<td>Yes</td>
<td>5</td>
</tr>
</tbody>
</table>
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:

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,

Semantics $\not\equiv$ Scalability $\Perp$

Consider progress on schema reasoning:

<table>
<thead>
<tr>
<th>Year</th>
<th>$O$-size</th>
<th>Complete</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>3,000</td>
<td>No</td>
<td>$10^5$</td>
</tr>
<tr>
<td>1998</td>
<td>3,000</td>
<td>Yes</td>
<td>300</td>
</tr>
<tr>
<td>2005</td>
<td>30,000</td>
<td>Yes</td>
<td>30</td>
</tr>
<tr>
<td>2010</td>
<td>400,000</td>
<td>Yes</td>
<td>5</td>
</tr>
</tbody>
</table>
Acknowledgements
References

Thank you for listening

Any questions?