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

Semantics $\sqcap$ Scalability $\models \bot$?
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
Semantic Technologies

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

[Logos of various semantic technologies]
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. eg:organisation
   - http://...rdf-syntax-ns/#type
2. eg:w3c
   - eg:worksfor
     - http://www.w3.org/People/EN/contact#me
3. eg:hq
   - W3C
4. eg:organisation
   - http://...fullName
5. eg:hq
   - Eric Miller
     - http://www.w3.org/2000/10/swp/pim/contact#mailbox
9. eg:organisation
   - Dr.
     - http://www.w3.org/2000/10/swp/pim/contact#personalTitle
   - http://www.w3.org/2000/10/swp/pim/contact#fullName
### How Does it Work?

1. **Standardised language for exchanging data**

   ```
<table>
<thead>
<tr>
<th>Triple</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
</tr>
<tr>
<td>em1234</td>
</tr>
<tr>
<td>em1234</td>
</tr>
<tr>
<td>em1234</td>
</tr>
<tr>
<td>em1234</td>
</tr>
<tr>
<td>em1234</td>
</tr>
<tr>
<td>w3c</td>
</tr>
<tr>
<td>w3c</td>
</tr>
<tr>
<td>w3c</td>
</tr>
<tr>
<td>...</td>
</tr>
</tbody>
</table>
   ```
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 provide a flexible naming scheme
   - Set of triples can be viewed as a graph

### How Does it Work?

<table>
<thead>
<tr>
<th>PERSON</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>NAME</td>
<td>TITLE</td>
<td>MAILBOX</td>
<td>WORKSFOR</td>
</tr>
<tr>
<td>em1234</td>
<td>“Eric Miller”</td>
<td>“Dr”</td>
<td><a href="mailto:em@w3.org">mailto:em@w3.org</a></td>
<td>w3c</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

| ORGANISATION | | |
|--------------|---|
| ID  | NAME    | HQ |
| w3c | “W3C” | Boston |
| ... | ... | ... |
How Does it Work?

Standardised language for exchanging vocabularies/schemas

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

```
Heart ⊆ MuscularOrgan ⊆
∃isPartOf.CirculatorySystem
HeartDisease ⊆ Disease
∃affects.Heart
VascularDisease ⊆ Disease
∃affects.(∃isPartOf.CirculatorySystem)
```
How Does it Work?

3. **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(\(?y,Boston\)\))
How Does it Work?

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

Patients suffering from vascular disease

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

Patients suffering from vascular disease

Q(p) ← Patient(p) \·
    suffersFrom(p, VascularDisease)
How Does it Work?

Patients suffering from vascular disease

\[
Q(?p) \leftarrow \text{Patient}(?p) \land \text{suffersFrom}(?p, \text{VascularDisease})
\]
How Does it Work?

Patients suffering from vascular disease

\[
\begin{align*}
\text{John} & \text{ rdf:type Patient} \\
\text{John suffersFrom d1} \\
\text{d1 rdf:type HeartDisease} \\
\text{Heart} & \text{MuscularOrgan} \quad \text{isPartOf.CirculatorySystem} \\
\text{HeartDisease} & \text{Disease} \quad \text{affects.Heart} \\
\text{VascularDisease} & \text{Disease} \quad \text{affects.(isPartOf.CirculatorySystem)}
\end{align*}
\]
How Does it Work?

Patients suffering from vascular disease

Q(p) ← Patient(p) ∧ suffersFrom(p, VascularDisease)

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

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

Patients suffering from vascular disease

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

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

How is heart disease a kind of vascular disease?

Q() ← subClassOf(HeartDisease, VascularDisease)
How Does it Work?

Is heart disease a kind of vascular disease?

\[
\text{Q()} \leftarrow \text{subClassOf(HeartDisease, VascularDisease)}
\]
How Does it Work?

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

\[
\begin{align*}
\text{Heart} & \sqsubseteq \text{MuscularOrgan} \\
& \sqsubseteq \text{IsPartOf.CirculatorySystem} \\
\text{HeartDisease} & \sqsubseteq \text{Disease} \\
& \sqsubseteq \text{Affects.Hear}t \\
\text{VascularDisease} & \sqsubseteq \text{Disease} \\
& \sqsubseteq \text{Affects.(IsPartOf.CirculatorySystem)}
\end{align*}
\]
How Does it Work?

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

Heart \text{\texttt{\equiv}} \text{MuscularOrgan}
\text{\texttt{\equiv}} \exists \text{PartOf.CirculatorySystem}

HeartDisease \text{\texttt{\equiv}} \text{Disease}
\text{\texttt{\equiv}} \exists \text{affects.Heart}

VascularDisease \text{\texttt{\equiv}} \text{Disease}
\text{\texttt{\equiv}} \exists \text{affects.} \exists \text{PartOf.CirculatorySystem}

Why?

Heart \Rightarrow \exists \text{PartOf.CirculatorySystem}, \ldots
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: 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 $\rightsquigarrow$ Practice

- OWL based on description logic $\mathcal{SROIQ}$
Theory $\leftrightarrow$ 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 $\iff$ 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 $\rightsquigarrow$ 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 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

1. Use EL “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>Logic</th>
<th>SNOMED CT</th>
<th>GALEN</th>
<th>FMA</th>
<th>GO</th>
</tr>
</thead>
<tbody>
<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>
</tbody>
</table>

| ELK (1 worker) | 13.15 | 1.33 | 0.44 | 0.20 |
| ELK (4 workers) | 5.02 | 0.77 | 0.39 | 0.19 |

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>#classes</td>
<td>19,145</td>
<td>1,728</td>
<td>70,576</td>
</tr>
<tr>
<td>#properties</td>
<td>82</td>
<td>145</td>
<td>189</td>
</tr>
<tr>
<td>#axioms</td>
<td>35,770</td>
<td>2,419</td>
<td>100,304</td>
</tr>
<tr>
<td>#⊆</td>
<td>$&gt; 10^{8}$</td>
<td>$&gt; 10^{6}$</td>
<td>$&gt; 10^{9}$</td>
</tr>
</tbody>
</table>

| 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?

<table>
<thead>
<tr>
<th></th>
<th>SNOMED CT</th>
<th>GALEN</th>
<th>FMA</th>
<th>GO</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logic</strong></td>
<td>$\mathcal{EL}$</td>
<td>$\mathcal{EL}$</td>
<td>$\mathcal{EL}$</td>
<td>$\mathcal{EL}$</td>
</tr>
<tr>
<td><strong>#classes</strong></td>
<td>315,489</td>
<td>23,136</td>
<td>78,977</td>
<td>19,468</td>
</tr>
<tr>
<td><strong>#properties</strong></td>
<td>58</td>
<td>950</td>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td><strong>#axioms</strong></td>
<td>430,844</td>
<td>36,547</td>
<td>121,712</td>
<td>28,897</td>
</tr>
<tr>
<td>$#\equiv$</td>
<td>$&gt;10^{11}$</td>
<td>$&gt;10^8$</td>
<td>$&gt;10^9$</td>
<td>$&gt;10^8$</td>
</tr>
<tr>
<td><strong>ELK (1 worker)</strong></td>
<td>13.15</td>
<td>1.33</td>
<td>0.44</td>
<td>0.20</td>
</tr>
<tr>
<td><strong>ELK (4 workers)</strong></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</th>
<th>Anat.</th>
<th>SWEET-P</th>
<th>NCI-2</th>
<th>DOLCE-P</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Logic</strong></td>
<td>$SHIF$</td>
<td>$SHOIN$</td>
<td>$ALCH$</td>
<td>$SHOIN$</td>
<td></td>
</tr>
<tr>
<td><strong>#classes</strong></td>
<td>19,145</td>
<td>1,728</td>
<td>70,576</td>
<td>118</td>
<td></td>
</tr>
<tr>
<td><strong>#properties</strong></td>
<td>82</td>
<td>145</td>
<td>189</td>
<td>264</td>
<td></td>
</tr>
<tr>
<td><strong>#axioms</strong></td>
<td>35,770</td>
<td>2,419</td>
<td>100,304</td>
<td>265</td>
<td></td>
</tr>
<tr>
<td>$#\equiv$</td>
<td>$&gt;10^8$</td>
<td>$&gt;10^6$</td>
<td>$&gt;10^9$</td>
<td>$&gt;10^4$</td>
<td></td>
</tr>
<tr>
<td><strong>HermiT</strong></td>
<td>11.2</td>
<td>11.2</td>
<td>—</td>
<td>105.1</td>
<td></td>
</tr>
<tr>
<td><strong>Pellet</strong></td>
<td>87.2</td>
<td>—</td>
<td>172.0</td>
<td>105.1</td>
<td></td>
</tr>
<tr>
<td><strong>FaCT++</strong></td>
<td>22.9</td>
<td>0.2</td>
<td>60.7</td>
<td>—</td>
<td></td>
</tr>
</tbody>
</table>
# Schema Reasoning — Solved Problem?

<table>
<thead>
<tr>
<th></th>
<th>SNOMED CT</th>
<th>GALEN</th>
<th>FMA</th>
<th>GO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic</td>
<td>$\mathcal{E}\mathcal{L}$</td>
<td>$\mathcal{E}\mathcal{L}$</td>
<td>$\mathcal{E}\mathcal{L}$</td>
<td>$\mathcal{E}\mathcal{L}$</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>#|$\subseteq$</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>$\text{SHIF}$</td>
<td>$\text{SHOIN}$</td>
<td>$\text{ALCH}$</td>
<td>$\text{SHOIN}$</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>#|$\subseteq$</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)

Restrictions may result in Byzantine/incorrect modelling

- e.g., in SNOMED, groin modelled as part of abdomen and of leg

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

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Classif. time (seconds)</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HermitT</td>
<td>MORe</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>total</td>
<td>HermitT</td>
<td>ELK</td>
<td></td>
</tr>
<tr>
<td>GO</td>
<td>7.1</td>
<td>2.2 (↓69.0%)</td>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>Gazetteer</td>
<td>838.1</td>
<td>28.2 (↓96.6%)</td>
<td>0</td>
<td>15.6</td>
</tr>
<tr>
<td>NCI</td>
<td>84.1</td>
<td>28.6 (↓66.0%)</td>
<td>15.8</td>
<td>3.3</td>
</tr>
<tr>
<td>Protein</td>
<td>11.4</td>
<td>2.9 (↓74.6%)</td>
<td>0.4</td>
<td>0.9</td>
</tr>
<tr>
<td>Biomodels</td>
<td>741.4</td>
<td>575.6 (↓22.4%)</td>
<td>540.1</td>
<td>2.6</td>
</tr>
<tr>
<td>cellCycle</td>
<td>–</td>
<td>13.9 (–)</td>
<td>&lt;0.1</td>
<td>4.9</td>
</tr>
<tr>
<td>NCI+MGED</td>
<td>1018.0</td>
<td>769.2 (↓24.4%)</td>
<td>605.5</td>
<td>2.4</td>
</tr>
<tr>
<td>NCI+CHEBI</td>
<td>116.6</td>
<td>34.0 (↓70.8%)</td>
<td>16.3</td>
<td>4.1</td>
</tr>
<tr>
<td>NCI+GO</td>
<td>110.0</td>
<td>37.6 (↓65.8%)</td>
<td>17.6</td>
<td>3.2</td>
</tr>
<tr>
<td>NCI+Mouse</td>
<td>93.7</td>
<td>31.0 (↓66.9%)</td>
<td>16.6</td>
<td>2.6</td>
</tr>
</tbody>
</table>
MORe Modular Reasoner

Currently improving our technique:

- Combining module extraction techniques with over approximation of ontology in a less expressive DL (EL/RL)

- Classification of over approximated ontology is fast, and is complete (but unsound) w.r.t. subsumption

- Significantly reduces number of subsumption checks delegated to powerful (but slower) reasoner
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 partial data materialisation + OBDA [2]
  - Datalog engine with (some form of) query rewriting [3]
  - Highly optimised ABox reasoners [4]


Various Approaches — Different Tradeoffs

➊ Use QL “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
Various Approaches — Different Tradeoffs

1. Use QL “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 QL ontology reasoners:
- E.g., Presto, 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 $\mathcal{M}$:
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** $Q \not\rightarrow Q'$ s.t. answering $Q'$ without $\mathcal{O}$ equivalent to answering $Q$ w.r.t. $\mathcal{O}$ for any dataset
- **Map** ontology queries $\not\rightarrow$ 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 $\mathcal{M}$:

- **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 $\mathcal{M}$ to rewrite $Q'$ into a DB query.

- **Evaluate** (SQL) query against DB.

\[ Q \xrightarrow{\text{Rewrite}} Q' \xrightarrow{\text{Map}} SQL \rightarrow DB \rightarrow \text{Ans} \]
Query Rewriting — Example

\[
\begin{align*}
\mathcal{O} & \left\{ \right. \\
\text{Doctor} & \subseteq \exists \text{treats. Patient} \\
\text{Consultant} & \subseteq \text{Doctor} \\
\mathcal{Q} & \left\{ \right. \\
Q(x) & \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \\
\mathcal{M} & \left\{ \right. \\
\text{Doctor} & \mapsto \text{SELECT Name FROM Doctor} \\
\text{Patient} & \mapsto \text{SELECT Name FROM Patient} \\
\text{treats} & \mapsto \text{SELECT DName, PName FROM Treats}
\end{align*}
\]
Query Rewriting — Example

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

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

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

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

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

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

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

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

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

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

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

\[ Q' \]

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

\[ M \]

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

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

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

\[ M \{ \]
\[ \text{Doctor} \rightarrow \text{SELECT Name FROM Doctor} \]
\[ \text{Patient} \rightarrow \text{SELECT Name FROM Patient} \]
\[ \text{treats} \rightarrow \text{SELECT DName, PName FROM Treats} \]

\[ \text{SQL} \{ \]
\[ \text{SELECT Name FROM Doctor UNION} \]
\[ \text{SELECT DName FROM Treats, Patient WHERE PName=Name} \]
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
   ▪ Relatively little work in this area to date

Various Approaches — Different Tradeoffs

2 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

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 $O$ and query $Q$: 
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
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

- **Evaluate** \( Q \) against DB'

---

![Diagram](image)

**Database (DB)** \( \rightarrow \) **Materialise** \( \rightarrow \) **DB'** \( \rightarrow \) **Answer (Ans)**
Materialisation — Example

\[
\emptyset \left\{ \right. \\
\text{Doctor} \equiv \exists \text{treats. Patient} \\
\text{Consultant} \sqsubseteq \text{Doctor}
\left. \right\}
\]

\[
\text{DB} \left\{ \\
treats(d_1, p_1) \\
\text{Patient}(p_1) \\
\text{Doctor}(d_2) \\
\text{Consultant}(c_1)
\right. \]
Materialisation — Example

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

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

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

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

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

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

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

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

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

\[ DB \{ \]
\[ \]
\[ \}

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

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

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

\[ O \left\{ \begin{array}{l}
\text{Doctor} \equiv \exists \text{treats}. \text{Patient} \\
\text{Consultant} \subseteq \text{Doctor}
\end{array} \right\} \]

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

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

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

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

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

\( O \) \{ 
Doctor \equiv \exists \text{treats}. \text{Patient} \\
\text{Consulatant} \subseteq \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) 
\}

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

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

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

\( Q_1 \mapsto \{d_2, d_1, c_1\} \)

\( Q_2 \mapsto \{d_1\} \)
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 incomplete for ontologies outside the profile:
  - Incompleteness not evenly distributed
  - Incompleteness difficult to measure via empirical testing
  - Can lead to unsoundness, e.g., via aggregation
- 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
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 $\langle A, \mathcal{Y} \rangle$ with $A$ an ABox and $\mathcal{Y}$ a query

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

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

\[ \emptyset \{ \text{Doctor} \equiv \exists \text{treats}\cdot \text{Patient} \}
\]

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

\[
\begin{align*}
\text{DB} \{ & \text{treats}(d_1, p_1) \\
& \text{Patient}(p_1) \\
& \text{Doctor}(d_2) \\
& \text{Consultant}(c_1) \\
\end{align*}
\]
Chase Materialisation — Example

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

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

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

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

$O \{ \begin{align*}
\text{Doctor} &\equiv \exists \text{treats}. \text{Patient} \\
\text{Consultant} &\subseteq \text{Doctor}
\end{align*} \}

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

$Q_1 \quad Q(x) \leftarrow \text{Doctor}(y) \quad \text{DB'} \begin{align*}
\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))
\end{align*} \] Skolems
Chase Materialisation — Example

\[
\begin{align*}
\mathcal{O} & \quad \{ \\
& \quad \text{Doctor} \equiv \exists \text{treats. Patient} \\
& \quad \text{Consultant} \sqsubseteq \text{Doctor} \\
\} \\
\mathcal{DB} & \quad \{ \\
& \quad \text{treats}(d_1, p_1) \\
& \quad \text{Patient}(p_1) \\
& \quad \text{Doctor}(d_2) \\
& \quad \text{Consultant}(c_1) \\
\} \\
\mathcal{Q}_1 & \quad Q(x) \leftarrow \text{Doctor}(y) \\
\mathcal{DB}' & \quad \{ \\
& \quad \text{treats}(d_1, p_1) \\
& \quad \text{Patient}(p_1) \\
& \quad \text{Doctor}(d_2) \\
& \quad \text{Consultant}(c_1) \\
& \quad \text{treats}(d_2, f(d_2)) \\
& \quad \text{Patient}(f(d_2)) \\
& \quad \text{treats}(c_1, f(c_1)) \\
& \quad \text{Patient}(f(c_1)) \\
& \quad \leadsto \\
& \quad \{d_2, d_1, c_1\} \\
\end{align*}
\]
Chase Materialisation — Example

\( O \) \{ 
    \text{Doctor} \equiv \exists \text{treats} \cdot \text{Patient} \\
    \text{Consultant} \sqsubseteq \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)) \\
\} \leadsto \{ d_2, d_1, c_1 \}

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

0 = \{ Doctor \equiv \exists \text{treats}.\text{Patient} \}
\quad \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{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) \quad \leadsto \quad \{d_2, d_1, c_1\}

Q_2 \quad Q(x) \leftarrow \text{treats}(x, y) \land \text{Patient}(y) \quad \leadsto \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$^{\pm,v}$ 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. \( O' \) 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
  - Delineates 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

Performance on LUBM

The graph shows the querying time (ms) and materialisation time (s) against the number of universities. The graph includes different lines for materialisation_lower, materialisation_upper, standard_lower, standard_upper, generated_lower, and generated_upper.
Performance on LUBM

≈ $10^8$ triples
Performance on LUBM

>7,000s using HermiT

\[ \approx 10^8 \text{ triples} \]
Performance on LUBM

$10^6$–$10^7$ answer tuples

$>7,000$ s using Hermit

$\approx 10^8$ triples
Performance on UOBM

The graph shows the querying time (in milliseconds) and materialisation time (in seconds) as a function of the number of universities. The lines represent different categories:

- `materialisation_lower`
- `materialisation_upper`
- `standard_lower`
- `standard_upper`
- `generated_lower`
- `generated_upper`

The x-axis represents the number of universities, ranging from 1 to 100. The y-axis for querying time is labeled in milliseconds, ranging from 100 to 100,000. The y-axis for materialisation time is labeled in seconds, also ranging from 100 to 100,000.
Performance on UOBM

\[ \approx 2 \times 10^7 \text{ triples} \]
Performance on UOBM

> $10^6$ answer tuples

$\approx 2 \times 10^7$ triples
Gap Between LB and UB

<table>
<thead>
<tr>
<th>Number of Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-0.1</td>
</tr>
<tr>
<td>0.1-0.2</td>
</tr>
<tr>
<td>0.2-0.3</td>
</tr>
<tr>
<td>0.3-0.4</td>
</tr>
<tr>
<td>0.4-0.5</td>
</tr>
<tr>
<td>0.9-1.0</td>
</tr>
<tr>
<td>1.0-2.0</td>
</tr>
<tr>
<td>2.0-3.0</td>
</tr>
<tr>
<td>3.0-4.0</td>
</tr>
<tr>
<td>4.0-5.0</td>
</tr>
<tr>
<td>5.0-6.0</td>
</tr>
<tr>
<td>10-14</td>
</tr>
</tbody>
</table>

|$\frac{\text{UB}}{\text{LB}}$|
Scalability: data storage

- **Choice of architectures:**
  - **distributed memory** – highly scalable, but reasoning is problematical
  - **single machine + secondary storage** – highly scalable, but access slow
  - **single machine + in memory storage** – fast, but limited scalability

- **In memory, fully indexed, efficient but compressed data structures:**
  - loading speed: 0.5 million triples per second
  - £4,000 desktop with 128 GB can store more than $2 \times 10^9$ triples
    - LUBM1000: 34b/t (48b/t including IRIs)
    - Claros: 40b/t (64b/t including IRIs)
Scalability: parallelization

- **Parallel data storage** tasks:
  - import, materialization, querying

- **Map reduce** approaches not well suited to reasoning
  - high communication overhead
  - redundant computation
  - query answering over (distributed) materialized data is problematical

- **Multiple processors with shared memory architecture**
  - no communication overhead
  - linear Datalog fragment theoretically parallelizable
  - data-driven parallel evaluation of general Datalog possible
    (parallelization depends on the data)
Scalability: parallel materialization

- **Materialization**: derivation stage + committing stage
Scalability: parallel materialization

- **Materialization**: derivation stage + committing stage
- Derivation stage: A prototype achieves 90% parallelization
- Committing stage (work in progress)
  - bottleneck: locking of hash tables
  - solution: fine-grained locking (striped hash tables)
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>
</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**:

<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>
</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**:

<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>
</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:

<table>
<thead>
<tr>
<th>Year</th>
<th>$\mathcal{O}$-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!**

<table>
<thead>
<tr>
<th>Year</th>
<th>Size</th>
<th>3000</th>
<th>No</th>
<th>10^4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>3,000</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1998</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>30,000</td>
<td>Yes</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>2010</td>
<td>400,000</td>
<td>Yes</td>
<td>5</td>
<td></td>
</tr>
</tbody>
</table>
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

Consider progress on schema reasoning:

<table>
<thead>
<tr>
<th>Year</th>
<th>$\mathcal{O}$-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>
References

Thank you for listening

Any questions?