CLASSIFYING ELH ONTOLOGIES IN SQL DATABASES

Vincent Delaitre and Yevgeny Kazakov
(Presented by Rob Shearer)

Oxford University Computing Laboratory

October 24, 2009
OUTLINE

1 INTRODUCTION

2 PROCEDURE OUTLINE

3 PROBLEMS AND SOLUTIONS

4 RESULTS
**ELH AND OWL 2 EL**

<table>
<thead>
<tr>
<th>OWL 2 Syntax</th>
<th>DL Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class expressions:</strong></td>
<td></td>
</tr>
<tr>
<td><code>ObjectIntersectionOf(C D)</code></td>
<td><code>C ∩ D</code></td>
</tr>
<tr>
<td><code>ObjectSomeValuesFrom(r C)</code></td>
<td><code>∃r.C</code></td>
</tr>
<tr>
<td><strong>Axioms:</strong></td>
<td></td>
</tr>
<tr>
<td><code>SubClassOf(C D)</code></td>
<td><code>C ⊑ D</code></td>
</tr>
<tr>
<td><code>EquivalentClasses(C D)</code></td>
<td><code>C ≡ D</code></td>
</tr>
<tr>
<td><code>SubObjectPropertyOf(r s)</code></td>
<td><code>r ⊑ s</code></td>
</tr>
</tbody>
</table>

- **ELH** is a simple sub-fragment of **OWL 2 EL**
- Has a very simple polynomial-time classification procedure ([Baader et al.,IJCAI 2003,2005](#))
- Sufficiently expressive for many ontologies such as SNOMED, FMA, NCI, GO and large part of GALEN
- Has a potential of scaling to even larger ontologies
Are we Ready for Ontologies with Millions of Classes?

- **SNOMED CT** contains over 300,000 classes—probably the largest ontology available so far.
- Can be classified in minutes using many existing reasoners:

<table>
<thead>
<tr>
<th>Reasoner</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEL</td>
<td>21min.42s.</td>
</tr>
<tr>
<td>FaCT++</td>
<td>16min.05s.</td>
</tr>
<tr>
<td>RACER</td>
<td>19min.30s.</td>
</tr>
<tr>
<td>SNOROCKET</td>
<td>1min.06s.</td>
</tr>
<tr>
<td>CB</td>
<td>0min.45s.</td>
</tr>
</tbody>
</table>
ARE WE READY FOR ONTOLOGIES WITH MILLIONS OF CLASSES?

- SNOMED CT contains over 300,000 classes—probably the largest ontology available so far.
- Can be classified in minutes using many existing reasoners:

<table>
<thead>
<tr>
<th>Reasoner</th>
<th>Time</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>CEL</td>
<td>21min.42s.</td>
<td>700MB*</td>
</tr>
<tr>
<td>FaCT++</td>
<td>16min.05s.</td>
<td>320MB*</td>
</tr>
<tr>
<td>RACER</td>
<td>19min.30s.</td>
<td>900MB</td>
</tr>
<tr>
<td>SNOROCKET</td>
<td>1min.06s.</td>
<td>2GB</td>
</tr>
<tr>
<td>CB</td>
<td>0min.45s.</td>
<td>400MB</td>
</tr>
</tbody>
</table>

- But memory consumption could be a problem for ontologies 10x larger.
SECONDARY MEMORY ONTOLOGY REASONING

The main idea: use a DBMS for processing of ontologies

Advantages:
1. Low main memory footprint
2. Persistence: can save / restore computations
3. Transactions and fault tolerance
4. Possible to adapt to multi-user environments

Disadvantage:
1. Slow (because of the secondary memory characteristics)
SECONDARY MEMORY ONTOLOGY REASONING

- The main idea: use a DBMS for processing of ontologies
- Advantages:
  1. Low main memory footprint
  2. Persistence: can save / restore computations
  3. Transactions and fault tolerance
  4. Possible to adapt to multi-user environments
- Disadvantage:
  1. Slow (because of the secondary memory characteristics)
- Our main results:
  - It is possible to classify \( \mathcal{ELH} \) ontologies in SQL databases
  - Naive approach has poor performance
  - Optimizations (caching) improve performance significantly
  - Able to classify SNOMED CT in 20min using 32MB of RAM.

Vincent Delaitre and Yevgeny Kazakov (Rob Shearer)
(Un)Related Works

- Conjunctive query answering in EL using relational databases [Lutz, Toman, Wolter, IJCAI 2009]
  - large instance data
  - medium-size schema (60,000 classes)
  - main focus is on query response

- “DB-backed” module in the IBM SHER system
  - Uses a Datalog engine
  - Presumably can work with EL+ ontologies
  - Cannot classify SNOMED CT(?)

- RDF databases:
  - Can query large triple stores
  - Can use custom rules
  - Cannot classify OWL ontologies(?)
Outline

1. Introduction

2. Procedure Outline

3. Problems and Solutions

4. Results
**ELH Classification Procedure**

1. **Normalization** to simple axioms of five forms:

   1. \( A \sqsubseteq B \)
   2. \( A \cap B \sqsubseteq C \)
   3. \( A \sqsubseteq \exists r.B \)
   4. \( \exists r.B \sqsubseteq C \)
   5. \( r \sqsubseteq s \)
**ELH** Classification Procedure

1. **Normalization** to simple axioms of five forms:
   
   (1) \( A \sqsubseteq B \)  
   (2) \( A \cap B \sqsubseteq C \)  
   (3) \( A \sqsubseteq \exists r.B \)  
   (4) \( \exists r.B \sqsubseteq C \)  
   (5) \( r \sqsubseteq s \)

**Example**

\[
A \sqsubseteq \exists r.(B \cap C) \quad \leadsto
\]
**ELH Classification Procedure**

1. **Normalization** to simple axioms of five forms:
   
   (1) \( A \sqsubseteq B \)  
   (2) \( A \cap B \sqsubseteq C \)  
   (3) \( A \sqsubseteq \exists r \cdot B \)  
   (4) \( \exists r \cdot B \sqsubseteq C \)  
   (5) \( r \sqsubseteq s \)

**Example**

\[
A \sqsubseteq \exists r \cdot (B \cap C) \quad \leadsto \quad A \sqsubseteq \exists r \cdot D \quad D \sqsubseteq B \cap C
\]
**ELH Classification Procedure**

1. **Normalization** to simple axioms of five forms:
   
   (1) \( A \subseteq B \)  
   (2) \( A \cap B \subseteq C \)  
   (3) \( A \subseteq \exists r.B \)  
   (4) \( \exists r.B \subseteq C \)  
   (5) \( r \subseteq s \)

**Example**

\[
A \subseteq \exists r.(B \cap C) \quad \iff \quad A \subseteq \exists r.D \quad D \subseteq B \cap C
\]
**ELH Classification Procedure**

1. **Normalization** to simple axioms of five forms:
   
   1. \( A \sqsubseteq B \)
   2. \( A \cap B \sqsubseteq C \)
   3. \( A \sqsubseteq \exists r.B \)
   4. \( \exists r.B \sqsubseteq C \)
   5. \( r \sqsubseteq s \)

**Example**

\[
A \sqsubseteq \exists r.(B \cap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C
\]
**ELH Classification Procedure**

1. Normalization to simple axioms of five forms:
   
   (1) $A \sqsubseteq B$
   
   (2) $A \cap B \sqsubseteq C$
   
   (3) $A \sqsubseteq \exists r . B$
   
   (4) $\exists r . B \sqsubseteq C$
   
   (5) $r \sqsubseteq s$

2. Deriving consequences using the rules [Brandt, ECAI 2004]:

**Example**

\[
A \sqsubseteq \exists r . (B \cap C) \quad \leadsto \quad A \sqsubseteq \exists r . D \quad D \sqsubseteq B \quad D \sqsubseteq C
\]
ELH Classification Procedure

1. Normalization to simple axioms of five forms:
   (1) $A \subseteq B$
   (2) $A \cap B \subseteq C$
   (3) $A \subseteq \exists r . B$
   (4) $\exists r . B \subseteq C$
   (5) $r \subseteq s$

**Example**

$$A \subseteq \exists r . (B \cap C) \iff A \subseteq \exists r . D \quad D \subseteq B \quad D \subseteq C$$

2. Deriving consequences using the rules [Brandt, ECAI 2004]:

IR1: $A \subseteq A$

IR2: $A \subseteq \top$

(tautologies)
**ELH Classification Procedure**

1. **Normalization to simple axioms of five forms:**
   
   (1) $A \subseteq B$  
   (2) $A \cap B \subseteq C$  
   (3) $A \subseteq \exists r.B$  
   (4) $\exists r.B \subseteq C$  
   (5) $r \subseteq s$

2. **Deriving consequences using the rules** [Brandt, ECAI 2004]:

   - **IR1**  
     $A \subseteq A$
   
   - **IR2**  
     $A \subseteq \top$

   - **CR1**  
     $A \subseteq B : B \subseteq C \in \mathcal{O}$
   
   - **CR2**  
     $A \subseteq B \quad A \subseteq C : B \cap C \subseteq D \in \mathcal{O}$

**Example**

\[
A \subseteq \exists r.(B \cap C) \quad \leadsto \quad A \subseteq \exists r.D \quad D \subseteq B \quad D \subseteq C
\]
**ELH Classification Procedure**

1. Normalization to simple axioms of five forms:
   
   (1) $A \sqsubseteq B$
   
   (2) $A \sqcap B \sqsubseteq C$
   
   (3) $A \sqsubseteq \exists r . B$
   
   (4) $\exists r . B \sqsubseteq C$
   
   (5) $r \sqsubseteq s$

2. Deriving consequences using the rules [Brandt, ECAI 2004]:

   **IR1**
   
   $A \sqsubseteq A$

   **IR2**
   
   $A \sqsubseteq \top$

   **CR1**
   
   $A \sqsubseteq B$
   
   $A \sqsubseteq C$

   $A \sqsubseteq C \in \mathcal{O}$

   **CR2**
   
   $A \sqsubseteq B$
   
   $A \sqsubseteq C$

   $B \sqsubseteq C \sqsubseteq D \in \mathcal{O}$

   **CR3**
   
   $A \sqsubseteq B$

   $A \sqsubseteq \exists r . C$

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

   **CR4**
   
   $A \sqsubseteq \exists r . B$

   $A \sqsubseteq \exists s . B$

   $r \sqsubseteq s \in \mathcal{O}$

   **CR5**
   
   $A \sqsubseteq \exists r . B$

   $B \sqsubseteq C$

   $A \sqsubseteq D$

   $\exists r . C \sqsubseteq D \in \mathcal{O}$
**Database Organization**

**Example**

Heart ⊑ MuscularOrgan  (type 1)
Heart ⊑ ∃isPartOf.CirculatorySystem  (type 3)

- Use two tables to assign ids to classes and object properties

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
<tr>
<td>CirculatorySystem</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td>1</td>
</tr>
</tbody>
</table>

- Use five tables to store normalized axioms of each type

<table>
<thead>
<tr>
<th>ax_t1</th>
<th>ax_t2</th>
<th>ax_t3</th>
<th>ax_t4</th>
<th>ax_t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⊑ B</td>
<td>A ∩ B ⊑ C</td>
<td>A ⊑ ∃r.B</td>
<td>∃r.B ⊑ C</td>
<td>r ⊑ s</td>
</tr>
<tr>
<td>1 2</td>
<td>. .</td>
<td>1 1 3</td>
<td>. .</td>
<td>. .</td>
</tr>
</tbody>
</table>
**Completion Using SQL Queries**

<table>
<thead>
<tr>
<th>ax_t1</th>
<th>ax_t2</th>
<th>ax_t3</th>
<th>ax_t4</th>
<th>ax_t5</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \subseteq B$</td>
<td>$A \cap B \subseteq C$</td>
<td>$A \subseteq \exists r. B$</td>
<td>$\exists r. B \subseteq C$</td>
<td>$r \subseteq s$</td>
</tr>
<tr>
<td>1 2</td>
<td>2 5 1</td>
<td>1 1 3</td>
<td>3 4 5</td>
<td>1 2</td>
</tr>
</tbody>
</table>

- Use two tables to output the results of inferences:

<table>
<thead>
<tr>
<th>s_t1</th>
<th>s_t2</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \subseteq B$</td>
<td>$A \subseteq \exists r. B$</td>
</tr>
<tr>
<td>1 1</td>
<td>1 1 3</td>
</tr>
</tbody>
</table>

- Use SQL commands to perform inferences:

**IR1**

$$A \subseteq A$$

**INSERT** INTO s_t1
**SELECT** class.id, class.id;

**CR1**

$$A \subseteq B, B \subseteq C \in \mathcal{O}$$

**INSERT IGNORE** INTO s_t1
**SELECT** s_t1.A, ax_t1.C
**FROM** s_t1 JOIN ax_t1
**ON** s_t1.B = ax_t1.A;
OUTLINE

1 INTRODUCTION

2 PROCEDURE OUTLINE

3 PROBLEMS AND SOLUTIONS

4 RESULTS
**Problem 1: Assignment of the IDs**

**Example**

Heart ⊑ MuscularOrgan  
Heart ⊑ ∃isPartOf.CirculatorySystem

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A ⊑ B  
A ∩ B ⊑ C  
A ⊑ ∃r.B  
∃r.B ⊑ C  
 r ⊑ s
### Problem 1: Assignment of the IDs

**Example**

<table>
<thead>
<tr>
<th>Class</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object Property</th>
<th>ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td></td>
</tr>
</tbody>
</table>

\[ A \subseteq B \]
\[ A \cap B \subseteq C \]
\[ A \subseteq \exists r.B \]
\[ \exists r.B \subseteq C \]
\[ r \subseteq s \]
**Problem 1: Assignment of the IDs**

**Example**

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart ⊑ ∃isPartOf.CirculatorySystem</td>
<td></td>
</tr>
</tbody>
</table>

On-disc table lookup is too slow! Making a query for every occurrence of a class is impractical due to overheads (connection + parsing + transaction).
### Problem 1: Assignment of the IDs

#### Example

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\exists\text{isPartOf}\cdot\text{CirculatorySystem}$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \subseteq B$</th>
<th>$A \cap B \subseteq C$</th>
<th>$A \subseteq \exists r \cdot B$</th>
<th>$\exists r \cdot B \subseteq C$</th>
<th>$r \subseteq s$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Problem 1: Assignment of the IDs**

**Example**

Heart ⊑ MuscularOrgan

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
</tbody>
</table>

Heart ⊑ ∃isPartOf.CirculatorySystem

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
A \subseteq B & \quad A \cap B \subseteq C \\
1 & \quad 2 \\
A \subseteq \exists r. B & \quad \exists r. B \subseteq C \\
1 & \quad 1 \\
r \subseteq s & \\
\end{align*}
\]
**Problem 1: Assignment of the IDs**

**Example**

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
<tr>
<td>CirculatorySystem</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\begin{align*}
A & \subseteq B & A \cap B & \subseteq C & A & \subseteq \exists r \cdot B & \exists r \cdot B & \subseteq C & r & \subseteq s \\
1 & 2 & & & 1 & 1 & 3 & &
\end{align*}
\]
**Problem 1: Assignment of the IDs**

**Example**

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
<tr>
<td>CirculatorySystem</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td>1</td>
</tr>
</tbody>
</table>

- On-disc table lookup is **too slow**!
- Making a query for every occurrence of a class is impractical due to overheads (connection + parsing + transaction)
**Solution: In-memory Caching**

**Example**

Heart $\subseteq$ MuscularOrgan  
Heart $\subseteq \exists$ isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

<table>
<thead>
<tr>
<th>On-Disk</th>
<th>In-Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td><strong>class</strong></td>
</tr>
<tr>
<td>Heart</td>
<td>Heart</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>MuscularOrgan</td>
</tr>
<tr>
<td><strong>id</strong></td>
<td><strong>id</strong></td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>object property</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A \sqsubseteq B$</td>
<td>$A \sqsubseteq B$</td>
</tr>
<tr>
<td>1 2</td>
<td></td>
</tr>
</tbody>
</table>

Vincent Delaitre and Yevgeny Kazakov (Rob Shearer)
**SOLUTION: IN-MEMORY CACHING**

**EXAMPLE**

Heart ⊑ MuscularOrgan  
Heart ⊑ ∃isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

<table>
<thead>
<tr>
<th>On-Disk</th>
<th>In-Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>class</strong></td>
<td><strong>id</strong></td>
</tr>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property id</th>
<th><strong>object property id</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>A ⊑ B</td>
<td>A ⊑ ∃r.B</td>
</tr>
<tr>
<td>1 2</td>
<td>3</td>
</tr>
</tbody>
</table>
**Solution: In-memory Caching**

**Example**

Heart $\sqsubseteq$ MuscularOrgan

Heart $\sqsubseteq \exists$ isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids

**On-Disk**

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
</tbody>
</table>

**In-Memory**

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>object property</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>isPartOf</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \sqsubseteq B$</th>
<th>$A \sqsubseteq \exists r.B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>1 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$A \sqsubseteq B$</th>
<th>$A \sqsubseteq \exists r.B$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 4</td>
<td>3 4</td>
</tr>
</tbody>
</table>
**Solution: In-memory Caching**

**Example**

Heart (MuscularOrgan)  
Heart (exists isPartOf CirculatorySystem)

- Insert into in-memory tables with fresh ids

<table>
<thead>
<tr>
<th>On-Disk</th>
<th>In-Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>id</td>
</tr>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
<tr>
<td>object property id</td>
<td></td>
</tr>
<tr>
<td>isPartOf</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>class</th>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Heart</td>
<td>3</td>
</tr>
<tr>
<td>CirculatorySystem</td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A ⊆ B</th>
<th>A ⊇ ∃r. B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 2</td>
<td>3 4 5</td>
</tr>
</tbody>
</table>

Vincent Delaitre and Yevgeny Kazakov (Rob Shearer)  
Classifying ELH Ontologies in SQL Databases 13/19
**EXAMPLE**

Heart ⊆ MuscularOrgan  
Heart ⊆ ∃isPartOf.CirculatorySystem

- Insert into in-memory tables with fresh ids
- Resolve uniqueness of ids using SQL queries when the tables are large enough

<table>
<thead>
<tr>
<th>On-Disk</th>
<th>In-Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>class</td>
<td>id</td>
</tr>
<tr>
<td>Heart</td>
<td>1</td>
</tr>
<tr>
<td>MuscularOrgan</td>
<td>2</td>
</tr>
<tr>
<td>object property</td>
<td>id</td>
</tr>
<tr>
<td>isPartOf</td>
<td>4</td>
</tr>
</tbody>
</table>

\[ A \sqsubseteq B \quad A \sqsubseteq \exists r. B \]

\[ 1 \quad 2 \quad 3 \quad 4 \quad 5 \]
Problem 2: Slow Joins

CR1
$A \sqcup B \quad : \quad B \sqcup C \in \mathcal{O}$

\[
\begin{array}{c|c|c}
A & B & C \\
1 & 1 & 2 \\
2 & 2 & 3 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
A & B & C \\
1 & 2 & 3 \\
2 & 4 & 3 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
s_{t1} & ax_{t1} \\
A & B \\
1 & 1 \\
2 & 4 \\
3 & 2 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
s_{t1} & ax_{t1} \\
A & B \\
1 & 2 \\
2 & 3 \\
\end{array}
\]

- Repeated application of joins are necessary to compute the closure

\[
\begin{array}{c|c|c}
\text{INSERT IGNORE INTO } s_{t1} \\
\text{SELECT } s_{t1}.A, ax_{t1}.C \\
\text{FROM } s_{t1} \text{ JOIN } ax_{t1} \\
\text{ON } s_{t1}.B = ax_{t1}.A;
\end{array}
\]
**Problem 2: Slow Joins**

\[
A \sqsubseteq B \quad : \quad B \sqsubseteq C \in O
\]

\[
\text{INSERT IGNORE INTO } \text{s}_\text{t1} \\
\text{SELECT } \text{s}_\text{t1}.A, \text{ax}_\text{t1}.C \\
\text{FROM } \text{s}_\text{t1} \text{ JOIN } \text{ax}_\text{t1} \\
\text{ON } \text{s}_\text{t1}.B = \text{ax}_\text{t1}.A;
\]

- Repeated application of joins are necessary to compute the closure.
PROBLEM 2: SLOW JOINS

CR1: \[
\begin{align*}
A \sqsubseteq B & : B \sqsubseteq C \in \mathcal{O} \\
A \sqsubseteq C & 
\end{align*}
\]

INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;

Repeated application of joins are necessary to compute the closure

<table>
<thead>
<tr>
<th>A \sqsubseteq B</th>
<th>s_t1</th>
<th>ax_t1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 1</td>
<td></td>
<td>1 2</td>
</tr>
<tr>
<td>2 2</td>
<td></td>
<td>1 4</td>
</tr>
<tr>
<td>3 3</td>
<td></td>
<td>2 3</td>
</tr>
<tr>
<td>1 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>


**PROBLEM 2: SLOW JOINS**

CR1

\[
\frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}
\]

INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;

Repeated application of joins are necessary to compute the closure

Instead one can compute the closure for a part of the table in-memory
**Problem 2: Slow Joins**

CR1

\[ A \sqsubseteq B \quad A \sqsubseteq C : B \sqsubseteq C \in \mathcal{O} \]

```sql
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory
**PROBLEM 2: SLOW JOINS**

\[
\begin{align*}
A & \subseteq B \\
A & \subseteq C \\
\Rightarrow B & \subseteq C \subseteq O
\end{align*}
\]

CR1

\[
\begin{align*}
A & \subseteq B \\
A & \subseteq C \\
\Rightarrow B & \subseteq C \subseteq O
\end{align*}
\]

```
INSERT IGNORE INTO s_t1
SELECT s_t1.A, ax_t1.C
FROM s_t1 JOIN ax_t1
ON s_t1.B = ax_t1.A;
```

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory
**Problem 2: Slow Joins**

\[
\begin{align*}
A & \sqsubset B & B & \sqsubset C \in \mathcal{O} \\
A & \sqsubset C
\end{align*}
\]

\[
\begin{array}{c|c|c}
& A \sqsubset B & A \sqsubset C \\
\hline
s_{t1} & 1 & 1 \\
& 2 & 2 \\
& 3 & 3 \\
\hline
ax_{t1} & 1 & 2 \\
& 1 & 4 \\
& 2 & 3 \\
\end{array}
\]

- Repeated application of joins are necessary to compute the closure.
- Instead one can compute the closure for a part of the table in-memory.
- And output the result into the main table.

```sql
INSERT IGNORE INTO s_{t1}
SELECT s_{t1}.A, ax_{t1}.C
FROM s_{t1} JOIN ax_{t1}
ON s_{t1}.B = ax_{t1}.A;
```
PROBLEM 2: SLOW JOINS

\[
\begin{align*}
A \sqsubseteq B & : B \sqsubseteq C \in \mathcal{O} \\
A \sqsubseteq C
\end{align*}
\]

\[
\text{INSERT IGNORE INTO } s_{t1} \\
\text{SELECT } s_{t1}.A, ax_{t1}.C \\
\text{FROM } s_{t1} \text{ JOIN } ax_{t1} \\
\text{ON } s_{t1}.B = ax_{t1}.A;
\]

- Repeated application of joins are necessary to compute the closure
- Instead one can compute the closure for a part of the table in-memory
- And output the result into the main table
- Repeat similarly for the other parts
Problem 3: Transitive Reduction

To produce the taxonomy, the output table needs to be transitively reduced.

\[
\begin{array}{|c|c|}
\hline
s_{t1} & A \sqsubseteq B \\
\hline
1 & 1 \\
2 & 2 \\
3 & 3 \\
1 & 2 \\
2 & 3 \\
1 & 3 \\
\hline
\end{array}
\]
To produce the taxonomy, the output table needs to be transitively reduced.

Can be done using one self join and marking the result as non-direct subsumptions.
Problem 3: Transitive Reduction

- To produce the taxonomy, the output table needs to be transitively reduced.
- Can be done using one self join and marking the result as non-direct subsumptions.
- This results in many on-disk updates since the number of non-direct subsumptions is large.

<table>
<thead>
<tr>
<th></th>
<th>A ⊑ B</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>
Problem 3: Transitive Reduction

To produce the taxonomy, the output table needs to be transitive reduced.

Can be done using one self join and marking the result as non-direct subsumptions.

This results in many on-disk updates since the number of non-direct subsumptions is large.

Instead, transitive reduction can be performed for parts of the table in-memory, marking only direct subsumptions on the disk.
To produce the taxonomy, the output table needs to be transitively reduced.

Can be done using one self join and marking the result as non-direct subsumptions.

This results in many on-disk updates since the number of non-direct subsumptions is large.

Instead, transitive reduction can be performed for parts of the table in-memory, marking only direct subsumptions on the disk.
Problem 3: Transitive Reduction

- To produce the taxonomy, the output table needs to be transitivity reduced.
- Can be done using one self join and marking the result as non-direct subsumptions.
- This results in many on-disk updates since the number of non-direct subsumptions is large.
- Instead, transitive reduction can be performed for parts of the table in-memory, marking only direct subsumptions on the disk.
Results

OUTLINE

1 INTRODUCTION

2 PROCEDURE OUTLINE

3 PROBLEMS AND SOLUTIONS

4 RESULTS
## Timings for Different Stages
(time in seconds)

<table>
<thead>
<tr>
<th>Action</th>
<th>NCI</th>
<th>GO</th>
<th>Galen^-</th>
<th>Snomed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loading/Preprocessing</td>
<td>17.85</td>
<td>5.99</td>
<td>23.41</td>
<td>127.51</td>
</tr>
<tr>
<td>Completion</td>
<td>5.78</td>
<td>7.29</td>
<td>53.13</td>
<td>783.30</td>
</tr>
<tr>
<td>Transitive reduction</td>
<td>10.32</td>
<td>6.10</td>
<td>21.44</td>
<td>249.23</td>
</tr>
<tr>
<td>Formating output</td>
<td>1.56</td>
<td>0.98</td>
<td>2.88</td>
<td>23.76</td>
</tr>
<tr>
<td>Total</td>
<td>35.51</td>
<td>20.36</td>
<td>100.86</td>
<td>1183.80</td>
</tr>
</tbody>
</table>

- **NCI** ([www.cancer.gov](www.cancer.gov)) contains 27,652 classes
- **GO** ([www.geneontology.org](www.geneontology.org)) contains 20,465 classes
- **Galen^-** ([www.co-ode.org/galen](www.co-ode.org/galen)) contains 23,136 classes (functionality, inverses, and transitivity removed)
- **Snomed** ([www.ihtsdo.org](www.ihtsdo.org)) contains 315,489 classes

Available at: [db-reasoner.googlecode.com](db-reasoner.googlecode.com)
# Results

## Comparison with In-Memory Reasoners (Time in Seconds)

<table>
<thead>
<tr>
<th>Reasoner</th>
<th>NCI</th>
<th>GO</th>
<th>Galen⁻</th>
<th>Snomed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CB</td>
<td>7.64</td>
<td>1.23</td>
<td>3.36</td>
<td>45.17</td>
</tr>
<tr>
<td>CEL v.1.0</td>
<td>3.60</td>
<td>1.02</td>
<td>169.23</td>
<td>1302.18</td>
</tr>
<tr>
<td>FaCT++ v.1.3.0</td>
<td>4.60</td>
<td>10.50</td>
<td>—</td>
<td>965.84</td>
</tr>
<tr>
<td>HermiT v.0.9.3</td>
<td>70.23</td>
<td>92.76</td>
<td>—</td>
<td>—</td>
</tr>
<tr>
<td>DB</td>
<td>35.51</td>
<td>20.36</td>
<td>100.86</td>
<td>1183.80</td>
</tr>
</tbody>
</table>

- **CB** ([cb-reasoner.googlecode.com](cb-reasoner.googlecode.com))
- **CEL** ([lat.inf.tu-dresden.de/systems/cel/](lat.inf.tu-dresden.de/systems/cel/))
- **FaCT++** ([owl.man.ac.uk/factplusplus](owl.man.ac.uk/factplusplus))
- **HermiT** ([hermit-reasoner.com](hermit-reasoner.com))
CONCLUSIONS

- **ELH** classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
Results

**Conclusions**

- **$\mathcal{ELH}$** classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
- Does it scale to millions of classes? –Not on a laptop:
  - Snomed x 1 = 20min
  - Snomed x 5 = 4h.30min
  - Snomed x 10 = did not finish overnight

Future work

- Extension to **OWL2 EL**
- Tuning the DB engine / testing on a real DB server

Please be kind and not ask too difficult questions!

or send them to: yevgeny.kazakov@comlab.ox.ac.uk

Vincent Delaitre and Yevgeny Kazakov (Rob Shearer)
CONCLUSIONS

- \texttt{ELH} classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
- Does it scale to millions of classes? –Not on a laptop:
  - Snomed x 1 = 20min
  - Snomed x 5 = 4h.30min
  - Snomed x 10 = did not finish overnight

Future work

- Extension to \texttt{OWL 2 EL}
- Tuning the DB engine / testing on a real DB server
CONCLUSIONS

- **ELH** classification is implementable in SQL databases
- Not as simple as it might first seem
- Optimizations are achieved using in-memory processing
- Performance is comparable to existing in-memory reasoners
- Does it scale to millions of classes? –Not on a laptop:
  - Snomed x 1 = 20min
  - Snomed x 5 = 4h.30min
  - Snomed x 10 = did not finish overnight

Future work

- Extension to **OWL 2 EL**
- Tuning the DB engine / testing on a real DB server

Please be kind and not ask too difficult questions!

or send them to: yevgeny.kazakov@comlab.ox.ac.uk