CLASSIFYING \mathcal{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

OWL 2 Syntax	DL Syntax	
Class expressions:		
ObjectIntersectionOf(C D)	$C\sqcap D$	
ObjectSomeValuesFrom(r C)	$\exists r.C$	
Axioms:		
SubClassOf(C D)	$C \sqsubseteq D$	
EquivalentClasses(CD)	$C \equiv D$	
SubObjectPropertyOf(r s)	$r \sqsubseteq s$	

- **ELH** is a simple sub-fragment of \mathcal{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:

	Time
CEL	21min.42s.
FaCT++	16min.05s.
RACER	19min.30s.
SNOROCKET	1min.06s.
CB	0min.45s.



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:

	Time	Memory
CEL	21min.42s.	700MB*
FaCT++	16min.05s.	320MB*
RACER	19min.30s.	900MB
SNOROCKET	1min.06s.	2GB
СВ	0min.45s.	400MB

But memory consumption could be a problem for ontologies 10x larger.





SECONDARY MEMORY ONOTLOGY REASONING

- The main idea: use a DBMS for processing of ontologies
- Advantages:
 - Low main memory footprint
 - Persistence: can save / restore computations
 - 3 Transactions and fault tolerance
 - 4 Possible to adapt to multi-user environments
- Disadvantage:
 - Slow (because of the secondary memory characteristics)





SECONDARY MEMORY ONOTLOGY REASONING

- The main idea: use a DBMS for processing of ontologies
- Advantages:
 - Low main memory footprint
 - Persistence: can save / restore computations
 - 3 Transactions and fault tolerance
 - 4 Possible to adapt to multi-user environments
- Disadvantage:
 - Slow (because of the secondary memory characteristics)
- Our main results:
 - lacktriangleright 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.



(UN)RELATED WORKS

- Conjunctive query answering in ££ 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 \mathcal{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



- 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) $r \sqsubseteq s$



- 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) $r \sqsubseteq s$

EXAMPLE

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto$$



- 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) $r \sqsubseteq s$

EXAMPLE

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.\underline{D} \quad \underline{D} \sqsubseteq B \sqcap C$$



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) $r \sqsubseteq s$

EXAMPLE

 $A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq \underline{B \sqcap C}$



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) $r \sqsubseteq s$

EXAMPLE

 $A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$



- 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) $r \sqsubseteq s$

EXAMPLE

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$



- 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) $r \sqsubseteq s$

EXAMPLE

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$

IR1
$$\frac{}{A \square A}$$
 IR2 $\frac{}{A \square \top}$ (tautologies)



- 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) $r \sqsubseteq s$

EXAMPLE

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$

IR1
$$A \sqsubseteq A$$
 IR2 $A \sqsubseteq T$ (tautologies)

$$\operatorname{CR1} \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \qquad \operatorname{CR2} \frac{A \sqsubseteq B \quad A \sqsubseteq C}{A \sqsubseteq D} : B \sqcap C \sqsubseteq D \in \mathcal{O}$$



- 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) $r \sqsubseteq s$

EXAMPLE

$$A \sqsubseteq \exists r.(B \sqcap C) \quad \leadsto \quad A \sqsubseteq \exists r.D \quad D \sqsubseteq B \quad D \sqsubseteq C$$

IR1
$$\frac{}{A \sqsubseteq A}$$
 IR2 $\frac{}{A \sqsubseteq \top}$ (tautologies)

$$\operatorname{CR1} \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \qquad \operatorname{CR2} \frac{A \sqsubseteq B \quad A \sqsubseteq C}{A \sqsubseteq D} : B \sqcap C \sqsubseteq D \in \mathcal{O}$$

$$\operatorname{CR3} \frac{A \sqsubseteq B}{A \sqsubseteq \exists r.C} : B \sqsubseteq \exists r.C \in \mathcal{O} \qquad \operatorname{CR4} \frac{A \sqsubseteq \exists r.B}{A \sqsubseteq \exists s.B} : r \sqsubseteq s \in \mathcal{O}$$

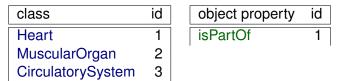
$$\text{CR5} \ \frac{A \sqsubseteq \exists r.B \quad B \sqsubseteq C}{A \sqsubseteq D} : \exists r.C \sqsubseteq D \in \mathcal{O}$$



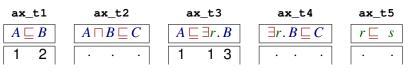
DATABASE ORGANIZATION

EXAMPLE	
Heart <u></u> MuscularOrgan	(type 1)
Heart <u>□</u> ∃isPartOf.CirculatorySystem	(type 3)

Use two tables to assign ids to classes and object properties



Use five tables to store normalized axioms of each type





COMPLETION USING SQL QUERIES

ax_t1ax_t2ax_t3ax_t4ax_t5
$$A \sqsubseteq B$$
 $A \sqcap B \sqsubseteq C$ $A \sqsubseteq \exists r.B$ $\exists r.B \sqsubseteq C$ $r \sqsubseteq s$ 122511134512

Use two tables to output the results of inferences:

Use SQL commands to perform inferences:

IR1
$$\overline{A \sqsubseteq A}$$
 INSERT INTO s_t1 SELECT class.id, class.id;

CR1 $\overline{A \sqsubseteq B} : B \sqsubseteq C \in \mathcal{O}$ 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;



OUTLINE

1 Introduction

2 PROCEDURE OUTLINE

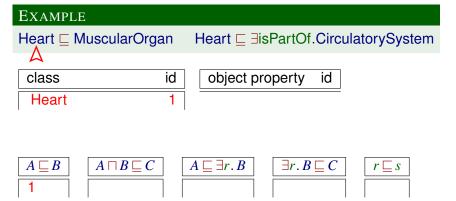
3 PROBLEMS AND SOLUTIONS

4 RESULTS

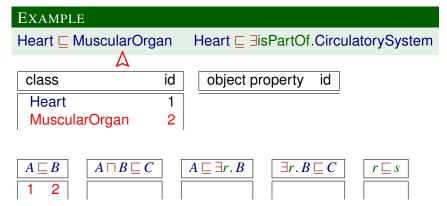


EXAMPLE	
Heart <u>□</u> MuscularOrgan	Heart <u>□</u> ∃isPartOf.CirculatorySystem
class id	object property id

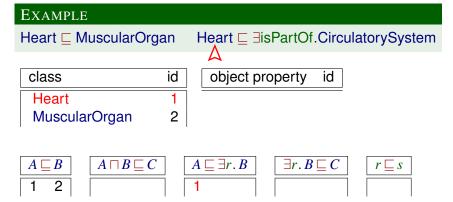








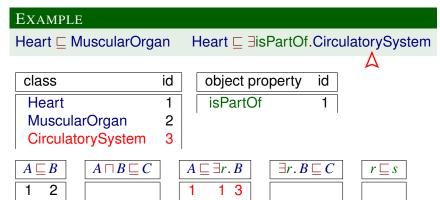






EXAMPLE		
Heart <u></u> MuscularOrga	an	Heart ⊑ ∃isPartOf.CirculatorySystem
class	id 1	object property id isPartOf 1
Muscular Organ $ \begin{array}{c c} A \sqsubseteq B & A \sqcap B \sqsubseteq C \\ \hline 1 & 2 & \end{array} $	2	$A \sqsubseteq \exists r. B \sqsubseteq C$ $r \sqsubseteq s$







EXAMPLE Heart ∃isPartOf.CirculatorySystem class id id object property Heart isPartOf MuscularOrgan 2 CirculatorySystem 3 $A \sqsubset B$ $A \sqcap B \sqsubseteq C$ $A \sqsubseteq \exists r. B$ $\exists r. B \sqsubseteq C$ $r \sqsubseteq s$ 2 3

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



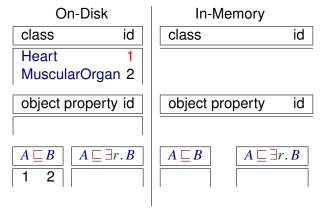
EXAMPLE

Heart

MuscularOrgan Heart

∃isPartOf.CirculatorySystem

Insert into in-memory tables with fresh ids



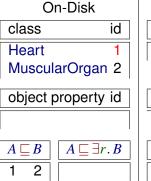


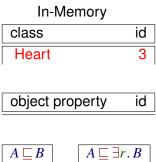
EXAMPLE

Heart ⊑ ∃isPartOf.CirculatorySystem

Δ

 Insert into in-memory tables with fresh ids





3



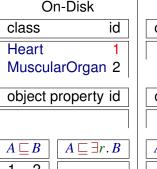
EXAMPLE

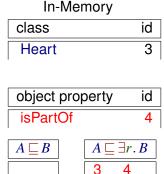
Heart

MuscularOrgan Heart

∃isPartOf.CirculatorySystem

 Insert into in-memory tables with fresh ids





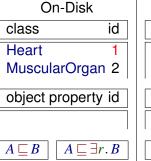


EXAMPLE

Heart <u>□</u> ∃isPartOf.CirculatorySystem

A Momony

 Insert into in-memory tables with fresh ids



in-iviemory			
class	id		
Heart	3		
CirculatorySystem 5			
object property id			
isPartOf	4		
$A \sqsubseteq B$	$A \sqsubseteq \exists r. B$		
	3 4 5		



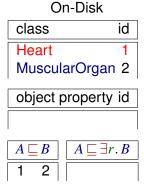
EXAMPLE

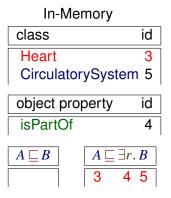
Heart

MuscularOrgan Heart

∃isPartOf.CirculatorySystem

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



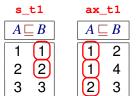




PROBLEM 2: SLOW JOINS

$$\begin{aligned} \operatorname{CR1} & \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ & \operatorname{INSERT} & \operatorname{IGNORE} & \operatorname{INTO} & \operatorname{s_t1} \\ & \operatorname{SELECT} & \operatorname{s_t1.A}, & \operatorname{ax_t1.C} \\ & \operatorname{FROM} & \operatorname{s_t1} & \operatorname{JOIN} & \operatorname{ax_t1} \\ & \operatorname{ON} & \operatorname{s} & \operatorname{t1.B} & = \operatorname{ax} & \operatorname{t1.A}; \end{aligned}$$

 Repeated application of joins are necessary to compute the closure

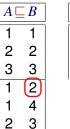




PROBLEM 2: SLOW JOINS

$$\begin{aligned} \operatorname{CR1} & \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ & \operatorname{INSERT} & \operatorname{IGNORE} & \operatorname{INTO} & \operatorname{s_t1} \\ & \operatorname{SELECT} & \operatorname{s_t1.A}, & \operatorname{ax_t1.C} \\ & \operatorname{FROM} & \operatorname{s_t1} & \operatorname{JOIN} & \operatorname{ax_t1} \\ & \operatorname{ON} & \operatorname{s} & \operatorname{t1.B} & = \operatorname{ax} & \operatorname{t1.A}; \end{aligned}$$

 Repeated application of joins are necessary to compute the closure



 s_t1

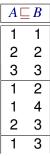


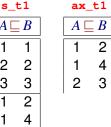


PROBLEM 2: SLOW JOINS

$$\begin{aligned} \operatorname{CR1} & \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ & \operatorname{INSERT} & \operatorname{IGNORE} & \operatorname{INTO} & \operatorname{s_t1} \\ & \operatorname{SELECT} & \operatorname{s_t1.A}, & \operatorname{ax_t1.C} \\ & \operatorname{FROM} & \operatorname{s_t1} & \operatorname{JOIN} & \operatorname{ax_t1} \\ & \operatorname{ON} & \operatorname{s} & \operatorname{t1.B} & = \operatorname{ax} & \operatorname{t1.A}; \end{aligned}$$

Repeated application of joins are necessary to compute the closure



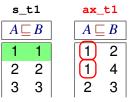




$$\begin{aligned} \operatorname{CR1} & \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ & \operatorname{INSERT} & \operatorname{IGNORE} & \operatorname{INTO} & \operatorname{s_t1} \\ & \operatorname{SELECT} & \operatorname{s_t1.A}, & \operatorname{ax_t1.C} \\ & \operatorname{FROM} & \operatorname{s_t1} & \operatorname{JOIN} & \operatorname{ax_t1} \end{aligned}$$

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



tmp

$$A \sqsubseteq B$$



$$\begin{aligned} \operatorname{CR1} \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ &\operatorname{INSERT \ IGNORE \ INTO \ s_t1} \\ &\operatorname{SELECT \ s_t1.A, \ ax_t1.C} \\ &\operatorname{FROM \ s_t1 \ JOIN \ ax_t1} \end{aligned}$$

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

s_t1		ax_	_t1
A	B	$A \sqsubseteq$	B
1	1	1	2
2	2	1	4
_	_		_

tmp





$$\label{eq:critical} \begin{aligned} \operatorname{CR1} \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ & \text{INSERT IGNORE INTO s_tl} \\ & \text{SELECT s_tl.A, ax_tl.c} \end{aligned}$$

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

s_t1		ax_	_t1	
$A \sqsubseteq B$			$A \sqsubseteq$	B
1	1		1	2
2	2		1	4
^	•		_	_

tmp

$A \sqsubseteq$	B
1	1
1	2
1	4
1	3



$$\label{eq:critical} \begin{aligned} \operatorname{CR1} \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O} \\ & \text{INSERT IGNORE INTO s_t1} \\ & \text{SELECT s_t1.A, ax_t1.C} \end{aligned}$$

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
- And output the result into the main table

s_	t1	ax_	
A	В	$A \sqsubseteq$	
1	1	1	
2	2	1	
2 3	2 3	2	
1	2	'	
1	4		
	_		

3



$$\operatorname{CR1} \frac{A \sqsubseteq B}{A \sqsubseteq C} : B \sqsubseteq C \in \mathcal{O}$$

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
- And output the result into the main table
- Repeat similarly for the other parts

 $\frac{\texttt{s_t1}}{A \square B}$

1

ax t1

 $A \sqsubset B$

3

tmp

 $A \sqsubseteq B$

2 2



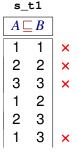
To produce the taxonomy, the output table needs to be transitively reduced

-
B
1
2
3
2
3
3

s t.1



- 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.





- 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

A	B	
1	1	×
2	2	×
3	3	×
1	2	
2	3	
1	3	×

s t1



- 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

A	B	
1	1	
2	2	
3	3	
1	2	
2	3	
1	3	
tmp		
4 -	- n	

s t.1

A	B
1	1
1	2
\ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \	2
- 1	3



- 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

_	_
$A \sqsubseteq$	В
1	1
2	2
3	3
1	2
2	3
1	3
tn	ıp
$A \sqsubseteq$	В

s t.1

3	×	

×



- 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

		_	
A	B		
1	1		
2	2		
2	3		
1	2	C	
2	3		
1	3		
tmp			
		1	

s t1

tr		
A	B	
1	1	×
1	2	0



OUTLINE

1 INTRODUCTION

2 PROCEDURE OUTLINE

3 PROBLEMS AND SOLUTIONS

4 RESULTS



TIMINGS FOR DIFFERENT STAGES (TIME IN SECONDS)

Action	NCI	GO	Galen-	Snomed
Loading/Preprocessing	17.85	5.99	23.41	127.51
Completion	5.78	7.29	53.13	783.30
Transitive reduction	10.32	6.10	21.44	249.23
Formating output	1.56	0.98	2.88	23.76
Total	35.51	20.36	100.86	1183.80

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

available at:

db-reasoner.googlecode.com





COMPARISON WITH IN-MEMORY REASONERS (TIME IN SECONDS)

Reasoner	NCI	GO	Galen-	Snomed
СВ	7.64	1.23	3.36	45.17
CEL v.1.0	3.60	1.02	169.23	1302.18
FaCT++ v.1.3.0	4.60	10.50		965.84
HermiT v.0.9.3	70.23	92.76	_	_
DB	35.51	20.36	100.86	1183.80

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



- ullet 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



- ullet 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



- ullet 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



- ullet 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

