Querying Semantic Big Data and Its Applications

Boris Motik

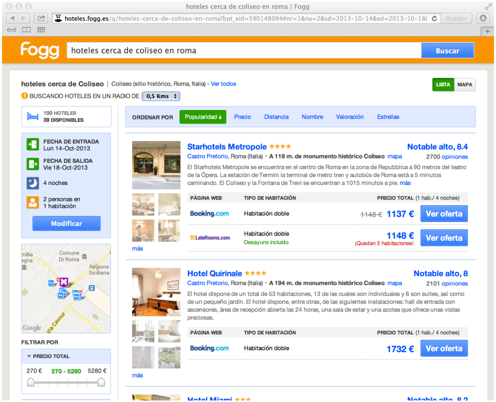
University of Oxford

November 16, 2015

Table of Contents

Table of Contents

Application: Search in Tourism (SkyScanner)



- Goal: search for hotels/flights/trips using natural language
- Need to represent large amounts of heterogeneous data
- Query for accommodation should include hotels, B&Bs, . . .

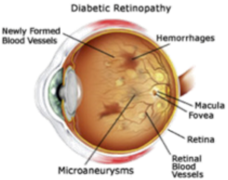
Application: Context-Aware Mobile Services (Samsung)

- Use sensors (WiFi, GPS, . . .) to identify the context
  - E.g., ‘at home’, ‘in a shop’, ‘with a friend’ . . .
- Adapt behaviour depending on the context
  - ‘If with a friend who has birthday, remind to congratulate’
- Declaratively describe contexts and adaptations
  - E.g., ‘If can see home Wifi, then context is “at home”’
- Interpret all rules in real-time using reasoning
- Main benefit: declarative, rather than procedural

Data Analysis in Healthcare (Kaiser Permanente)

- HEDIS[1] is a Performance Measure specification issued by NCQA[2]
  - E.g., all diabetic patients must have annual eye exams

- Meeting HEDIS standards is a requirement for government funded healthcare (Medicare)



- Checking/reporting is difficult and costly
  - Complex specifications & annual revisions
  - Disparate data sources
  - Ad hoc schemas including implicit information

- ⇒ Our solution: specify reporting rules declaratively (in datalog)
  - Easier creation, debugging, and maintenance

[1] Healthcare Effectiveness Data and Information Set
[2] National Committee for Quality Assurance

Information Integration in Gas & Oil (Statoil)

- Geologists & geophysicists use data from previous operations in nearby locations to develop stratigraphic models of unexplored areas
  - TBs of relational data
  - Diverse schemata
  - Spread over 1,000s of tables and multiple data bases
- Data Access
  - 900 geologists & geophysicists
  - 30–70% of time on data gathering
  - four-day turnaround for new queries
- Data Exploitation
  - Better use of experts time
  - Data analysis ‘most important factor’ for drilling success

Optique

Common Problem: Query Answering

OWL 2 DL — Language for Ontology Modelling

- Each ontology can be normalised to disjunctive existential rules:

$$\forall \vec{x}\vec{z}.\ \left[\varphi(\vec{x},\vec{z}) \rightarrow \exists \vec{y}_1.\psi_1(\vec{x},\vec{y}_1) \vee \ldots \vee \vec{y}_n.\psi_n(\vec{x},\vec{y}_n)\right]$$

  - φ and ψ_i are conjunctions of atoms
  - Predicates are unary (i.e., concepts), binary (i.e., roles), or $\approx$
  - Various structural restrictions ensure decidability

Conjunctive Query Answering

- Conjunctive queries: $Q(\vec{x}) \equiv \exists \vec{y}.\varphi(\vec{x},\vec{y})$
- Query answering: find all ground τ such that $\mathcal{O} \models Q(\vec{x})\tau$

OWL 2 DL Fragments

- OWL 2 RL — finite domain $\Rightarrow$ datalog query answering
- OWL 2 EL — polynomial subsumption (i.e., checking $\mathcal{O} \models \forall x.[A(x) \rightarrow B(x)]$)
- OWL 2 QL — data complexity of query answering in AC^0

Table of Contents

Goals of RDFox

- Develop techniques for materialisation of datalog programs on RDF data

Goals of RDFox

- Develop techniques for materialisation of datalog programs on RDF data

- Current trends in databases and knowledge-based systems:
  - Price of RAM keeps falling
    - 128 GB is routine, systems with 1 TB are emerging
    - In-memory databases: SAP's HANA, Oracle's TimesTen, YarcData's Urika
  - Materialisation is computationally intensive $\Rightarrow$ natural to parallelise
    - Mid-range laptops have 4 cores, servers with 16 cores are routine

Goals of RDFox

- Develop techniques for materialisation of datalog programs on RDF data in main-memory, multicore systems
  - Implemented in the RDFox system
  - `http://www.cs.ox.ac.uk/isg/tools/RDFox/`

- Current trends in databases and knowledge-based systems:
  - Price of RAM keeps falling
    - 128 GB is routine, systems with 1 TB are emerging
    - In-memory databases: SAP's HANA, Oracle's TimesTen, YarcData's Urika
  - Materialisation is computationally intensive $\Rightarrow$ natural to parallelise
    - Mid-range laptops have 4 cores, servers with 16 cores are routine

B. Motik, Y. Nenov, R. Piro, I. Horrocks, D. Olteanu: Parallel Materialisation of Datalog Programs in Centralised, Main-Memory RDF Systems. AAAI 2014
B. Motik, Y. Nenov, R. Piro, I. Horrocks.: Handling owl:sameAs via Rewriting. AAAI 2015

Existing Approaches to Parallel Materialisation

- Interquery parallelism: run independent rules in parallel
  - Degree of parallelism limited by the number of independent rules
  - $\Rightarrow$ does not distribute workload to cores evenly
- Intraquery parallelism
  - Partition rule instantiations to N threads
    - E.g., constrain the body of rules evaluated by thread i to (x mod $N = i$)
    - $\Rightarrow$ Static partitioning may not distribute workload well due to data skew
    - $\Rightarrow$ Dynamic partitioning may incur an overhead due to load balancing
  - Parallelise join computation
    - Hash-partition data into blocks, compute the join for each block independently
    - $\Rightarrow$ Hash tables keep being constantly recomputed
    - Sort-merge join requires constant data reordering
- Goal: distribute workload to threads evenly and with minimum overhead

Interleaving Querying with Updates

- Efficient query evaluation requires indexes
  - Crucial for elimination of duplicate triples $\Rightarrow$ ensures termination
  - Usually sorted (and clustered) to allow for merge joins
  - Hash indexes can also be used
  - Individual (i.e., not bulk) index updates are inefficient

- Materialisation interleaves . . .
  - . . . querying (during evaluation of rule bodies)
  - . . . updates (during updates of derived facts)

- $\Rightarrow$ Data storage should support indexes and efficient parallel updates

SOLUTION PART I: ALGORITHM

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)

$$A(x) \wedge R(x,y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery:

SOLUTION PART I: ALGORITHM

⇒ R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: A(a)

Solution Part I: Algorithm

R(a,b)
⇒ R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)

$$A(x) \wedge R(x,y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: A(a)

Solution Part I: Algorithm

R(a,b)
R(a,c)
⇒ R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: A(b)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
⇒ R(b,e)
A(a)
R(c,f)
R(c,g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: A(b)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
⇒ A(a)
R(c,f)
R(c,g)
A(b)
A(c)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: R(a,y)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
⇒ R(c,f)
R(c,g)
A(b)
A(c)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: A(c)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
⇒ R(c,g)
A(b)
A(c)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: A(c)

SOLUTION PART I: ALGORITHM

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)
⇒ A(b)
A(c)
A(d)
A(e)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: R(b,y)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)
A(b)
⇒ A(c)
A(d)
A(e)
A(f)
A(g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: R(c,y)

SOLUTION PART I: ALGORITHM

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)
A(b)
A(c)
⇒ A(d)
A(e)
A(f)
A(g)

$$A(x) \wedge R(x,y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: R(d,y)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)
A(b)
A(c)
A(d)
⇒ A(e)
A(f)
A(g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: R(e,y)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)
A(b)
A(c)
A(d)
A(e)
⇒ A(f)
A(g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

Current subquery: R(f,y)

Solution Part I: Algorithm

R(a,b)
R(a,c)
R(b,d)
R(b,e)
A(a)
R(c,f)
R(c,g)
A(b)
A(c)
A(d)
A(e)
A(f)
⇒ A(g)

$$A(x) \wedge R(x, y) \rightarrow A(y)$$

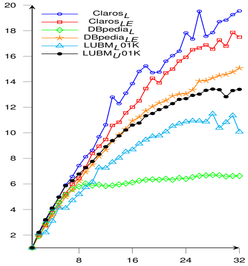
- For each fact:
  1. Match the fact to all body atoms to obtain subqueries
  2. Evaluate subqueries w.r.t. all previous facts
  3. Add results to the table

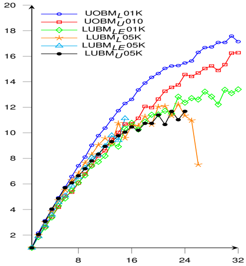
Current subquery: R(g,y)

Solution Part II: Data Indexing & Lock-Free Updates

- Lock-based programming
  - Main benefit: simplicity, easy to ensure linearisability
  - Main problem: susceptible to thread scheduling
    - A thread acquires a lock and goes to sleep $\Rightarrow$ block progress of all other threads
    - Can happen due to swapping, causes priority inversion
- Lock-free programming
  - At all time, at least one thread makes progress
  - Commonly implemented using compare-and-set: CAS(*loc*, *exp*, *new*)
    - Load the value stored of location *loc* into temporary variable *old*
    - Store *new* into location *loc* if *old* $=$ *exp*
    - Hardware ensures atomicity
  - A thread can wait indefinitely (e.g., CAS may keep failing)
  - (Unlike wait-free programming: each thread progresses after a fixed amount of time)
- Complete lock-freedom can be costly $\Rightarrow$ we resort to locks occasionally
  - 'Mostly' lock-free

Evaluation: Parallelisation Overhead and Speedup





- Small concurrency overhead; parallelisation pays off already with two threads
- Speedup of up to 13x with 16 physical cores
- Increases to 19x with 32 virtual cores

EVALUATION: ORACLE'S SPARC T5 (128/1024 CORES, 4 TB)

	LUBM-50K		Claros		DBpedia	
Threads	sec	speedup	sec	speedup	sec	speedup
import	6.8k	—	168	—	952	—
1	27.0k	1.0x	10.0k	1.0x	31.2k	1.0x
16	1.7k	15.7x	906.0	11.0x	3.0k	10.4x
32	1.1k	24.0x	583.3	17.1x	1.8k	17.5x
48	920.7	29.3x	450.8	22.2x	2.0k	16.0x
64	721.2	37.4x	374.9	26.7x	1.2k	25.8x
80	523.6	51.5x	384.1	26.0x	1.2k	26.7x
96	442.4	60.9x	364.3	27.4x	825	37.8x
112	400.6	67.3x	331.4	30.2x	1.3k	24.3x
128	387.4	69.6x	225.7	44.3x	697.9	44.7x
256	—	—	226.1	44.2x	684.0	45.7x
384	—	—	189.1	52.9x	546.2	57.2x
512	—	—	153.5	65.1x	431.8	72.3x
640	—	—	140.5	71.2x	393.4	79.4x
768	—	—	130.4	76.7x	366.2	85.3x
896	—	—	127.0	78.8x	364.9	86.6x
1024	—	—	124.9	80.1x	358.8	87.0x
size	B/trp	Triples	B/trp	Triples	B/trp	Triples
aft imp	124.1	6.7G	80.5	18.8M	58.4	112.7M
aft mat	101.0	9.2G	36.9	539.2M	39.0	1.5G
import rate	1.0M		112k		120k	
mat. rate	6.1M		4.2M		4.0M	

Incremental Materialisation Maintenance

- Common application scenario: continuous small changes in input data
- Incremental maintenance: update materialisation with minimal effort

Incremental Materialisation Maintenance

- Common application scenario: continuous small changes in input data
- Incremental maintenance: update materialisation with minimal effort

- State of the art (from the 90s):
  - the Counting algorithm
    - Basic variant applicable only to nonrecursive programs!
    - Extension to recursive programs rather complex
  - the Delete/Rederive (DRed) algorithm
    - Works for nonrecursive rules too
  - Unclear which algorithms is ‘better’
    - Complexity is the same
    - No empirical comparison thus far

Incremental Materialisation Maintenance

- Common application scenario: continuous small changes in input data
- Incremental maintenance: update materialisation with minimal effort

- State of the art (from the 90s):
  - the Counting algorithm
    - Basic variant applicable only to nonrecursive programs!
    - Extension to recursive programs rather complex
  - the Delete/Rederive (DRed) algorithm
    - Works for nonrecursive rules too
  - Unclear which algorithms is ‘better’
    - Complexity is the same
    - No empirical comparison thus far

- Our Forward/Backward/Forward (FBF) algorithm often outperforms DRed
  - Extensive empirical comparison with counting on the way

B. Motik, Y. Nenov, R. Piro, I. Horrocks.:
Incremental Update of Datalog Materialisation: the Backward/Forward Algorithm. AAAI 2015
Combining Rewriting and Incremental Materialisation Maintenance for Datalog Programs with Equality. IJCAI 2015

Table of Contents

OWL 2 EL

Example OWL 2 EL Ontology $\mathcal{O}$

$$A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)] \qquad B(x) \rightarrow \exists y.[S(x,y) \wedge A(y)]$$

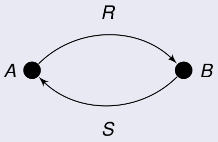
OWL 2 EL

Example OWL 2 EL Ontology $\mathcal{O}$

$$A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)] \qquad B(x) \rightarrow \exists y.[S(x,y) \wedge A(y)]$$

‘Folded’ Models

- Introduce one node for each concept
- Finite (polynomial) $\Rightarrow$ can be efficiently materialised using datalog
- Sufficient for concept subsumption



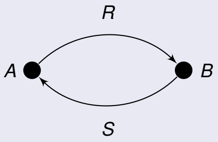
OWL 2 EL

Example OWL 2 EL Ontology $\mathcal{O}$

$$A(x) \rightarrow \exists y.[R(x,y) \wedge B(y)] \qquad B(x) \rightarrow \exists y.[S(x,y) \wedge A(y)]$$

'Folded' Models

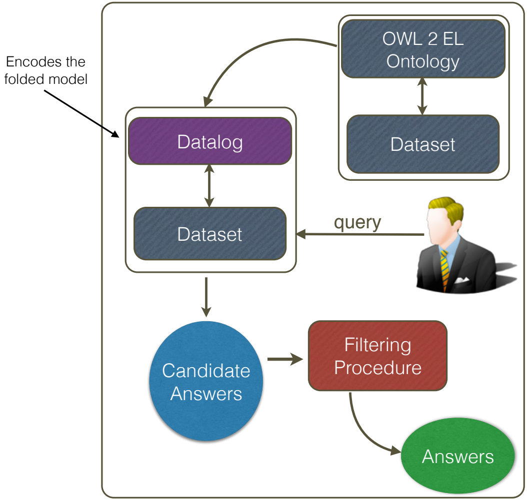
- Introduce one node for each concept
- Finite (polynomial) $\Rightarrow$ can be efficiently materialised using datalog
- Sufficient for concept subsumption



Query Answering Problems

- Evaluating a query in a folded model is unsound
- E.g., $Q \equiv \exists x, y.[R(x,y) \wedge S(y,x)]$
- Q is false over $\mathcal{O}$
- But, Q is true in the 'folded' model

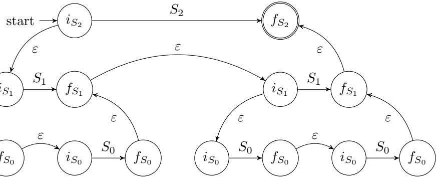
Combined Approach to Query Answering in OWL 2 EL



Open Problems in Known Approaches

1. Original combined approaches proposed for $\mathcal{ELH}$
   - Filtering implemented 'inside the query'
   - Missing features:
     - Complex role inclusions (e.g., $\mathit{parentOf}(x, y) \wedge \mathit{siblingOf}(y, z) \rightarrow \mathit{parentOf}(x, z)$)
     - Nominals (e.g., $\mathit{OxfordProf}(x) \rightarrow \mathit{worksAt}(x, \mathit{OxfordUni})$)
     - Reflexivity (e.g., $\mathit{Narcissist}(x) \rightarrow \mathit{loves}(x, x)$)

2. Existing query answering procedures are not optimal:
   - Regular complex role inclusions compiled to automata
   - $\Rightarrow$ Can incur exponential blowup
   - For example, $S_{i-1}(x, y) \wedge S_{i-1}(y, z) \rightarrow S_i(x, z)$ with $1 \leq i \leq 2$ produces



New Filtering Procedure for OWL 2 EL

- PSpace in case OWL 2 EL
  - We compile role inclusions into pushdown automata with bounded stack
  - ⇒ Tight upper complexity bound
- NP in case of transitivity
  - Worst-case optimal: checking candidate answer soundness is NP-hard
  - Optimised to reduce nondeterminism in common practical cases
- Polynomial in case no transitivity and no complex role inclusions
- ⇒ 'Pay-as-you-go' behaviour

G. Stefanoni, B. Motik: Answering Conjunctive Queries over EL Knowledge Bases with Transitive and Reflexive Roles. AAAI 2015
G. Stefanoni, B. Motik, M. Krötzsch, S. Rudolph: The Complexity of Answering Conjunctive and Navigational Queries over OWL 2 EL Knowledge Bases. JAIR
G. Stefanoni, B. Motik, I. Horrocks: Introducing Nominals to the Combined Query Answering Approaches for EL. AAAI 2013

Performance Evaluation

- KARMA: a prototype system based on RDFox

(a) LSTW results for queries that do not use transitive roles

	q_1^l				q_2^l				q_5^l				q_8^l				q_9^l				q_{10}^t			
	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N
L5	111.9K	4.0	0.009	0	3.6M	100	0.010	0	27.9K	0	0.003	0	9.6K	0	0.002	0	1.1K	0	0.003	0	3.2K	0	0.001	0
L10	223.5K	4.2	0.009	0	32.0M	100	0.009	0	57.4K	0	0.002	0	19.4K	0	0.002	0	2.2K	0	0.005	0	6.4K	0	0.001	0
L20	487.3K	4.3	0.006	0	170.3M	100	0.009	0	121.2K	0	0.002	0	41.2K	0	0.002	0	4.8K	0	0.007	0	13.7K	0	0.001	0

(b) LSTW results for queries that use transitive roles

	q_3^l				q_7^l				q_{12}^l				q_{13}^l				q_{14}^l				q_{15}^l			
	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N
L5	10	0	0.001	0	19K	0	2.845	5.8	73K	12	1.71	7.55	3K	0	0.01	0	157K	66	1.07	8.6	30K	63	2.44	10.9
L10	22	0	0.001	0	38K	0	2.808	5.8	149K	12	1.68	7.54	6K	0	0.01	0	603K	81	1.20	9.6	61K	63	2.44	10.9
L20	43	0	0.001	0	82K	0	2.800	5.8	313K	12	1.66	7.55	12K	0	0.01	0	2.6M	90	1.28	10.3	129K	63	2.44	10.9

(c) SEMINTEC results

	q_1^s				q_2^s				q_3^s				q_4^s				q_5^s				q_6^s				q_7^s				q_8^s			
	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N	C	U	F	N
SEM	7	0	0.001	0.0	53	0	0.01	0	16	0	0.125	0	12	0	0.001	0	31	0	0.096	0	838K	55	0.004	0	2.2K	0	0.006	0	13K	0	0.004	0

C: # candidate answers U: % of unsound answers F: avg. filtering time (ms) N: avg. # nondeterministic choices

- $\Rightarrow$ Approach is practical!

Table of Contents

Sources of Difficulty to Practical Query Answering

1. Handling existential variables in queries is a major complexity source
   - 2ExpTime-hard for even simple logics
   - Decidable for OWL 2 DL, but exact complexity unknown
   - Algorithms typically exhibit worst-case complexity on all inputs
   - $\Rightarrow$ Simplifying assumption: no existentially quantified variables
   - Sufficient for all applications known to us
2. No canonical model to evaluate queries
   - Practical reasoning provided by tableau algorithms $\Rightarrow$ only decision procedures
   - $\Rightarrow$ May require exponentially many algorithm runs
   - $\Rightarrow$ Goal-oriented search for answers very difficult
3. Tableau algorithms cannot handle large knowledge bases
   - Thousands of assertions at most
   - $\Rightarrow$ Nowhere near 'big data'

The PAGOdA Approach

1. Find the lower bound answer
   - E.g., answer Q w.r.t. the datalog part of the TBox
   - E.g., answer Q w.r.t. the OWL 2 EL part of the TBox
   - $\Rightarrow$ sound, but incomplete
   - Can be done efficiently using RDFox
   - Hope: retrieves the majority of answers in many practical cases

2. Find the upper bound
   - Replace existential variables with constants; replace $\vee$ with $\wedge$
   - $\Rightarrow$ complete, but unsound
   - Can be done efficiently using RDFox
   - Hope: (upper $\setminus$ lower) bound is small

3. For each answer in (upper $\setminus$ lower) bound:
   - Extract the relevant part of the ABox
   - Check the answer's validity using a sound & complete reasoner (e.g., HermiT)
   - Hope: the relevant ABox part is small

Y. Zhou, Y. Nenov, B. Cuenca Grau, I. Horrocks: Pay-As-You-Go OWL Query Answering Using a Triple Store. AAAI 2014
Y. Zhou, Y. Nenov, B. Cuenca Grau, I. Horrocks: Complete Query Answering over Horn Ontologies Using a Triple Store. ISWC 2013
Z. Zhou, B. Cuenca Grau, I. Horrocks, Z. Wu, J. Banerjee: Making the most of your triple store: query answering in OWL 2 using an RL reasoner. WWW 2013

PAGODA EXAMPLE (I)

TBOX

$$\mathit{worksFor}(x, z_1) \wedge \mathit{hasContract}(x, z_2) \wedge \mathit{Permanent}(z_2) \rightarrow \mathit{PermEmployee}(x)$$
$$\mathit{Employee}(x) \rightarrow \exists y.\mathit{worksFor}(x, y)$$

ABOX

$$\mathit{worksFor}(\mathit{peter}, \mathit{GSK}) \qquad \mathit{hasContract}(\mathit{peter}, c_1) \qquad \mathit{Permanent}(c_1)$$
$$\mathit{Employee}(\mathit{paul}) \qquad \mathit{hasContract}(\mathit{paul}, c_2) \qquad \mathit{Permanent}(c_2)$$

QUERY

$$Q(X) \equiv \mathit{PermEmployee}(x)$$

Answer: $\{\mathit{peter}, \mathit{paul}\}$

PAGODA EXAMPLE (II)

ABOX

$worksFor(peter, GSK)$ $hasContract(peter, c_1)$ $Permanent(c_1)$
$Employee(paul)$ $hasContract(paul, c_2)$ $Permanent(c_2)$

LOWER BOUND

$$worksFor(x, y_1) \wedge hasContract(y_2) \wedge Permanent(y_2) \rightarrow PermEmployee(x)$$

Answer: $\{peter\}$

LOWER BOUND

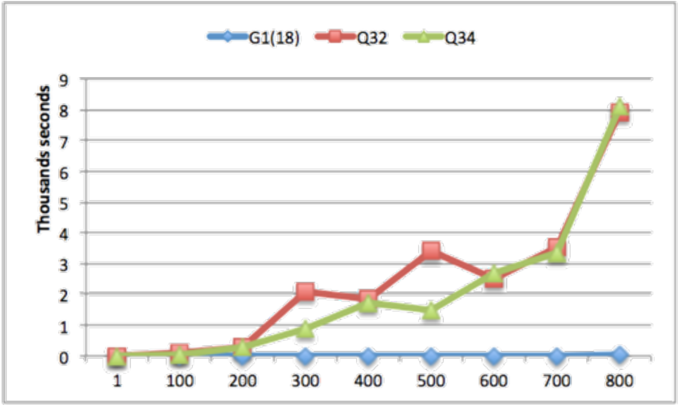
$$\begin{aligned} worksFor(x, y_1) \wedge hasContract(y_2) \wedge Permanent(y_2) &\rightarrow PermEmployee(x) \\ Employee(x) &\rightarrow worksFor(x, SK_1) \end{aligned}$$

Answer: $\{peter, paul\}$

RELEVANT ABOX PART FOR *paul*

$Employee(paul)$ $hasContract(paul, c_2)$ $Permanent(c_2)$

Performance Evaluation



LUBM query processing

Table of Contents

Research Directions

- Increase capacity of RDFox using a shared-nothing cluster
  - Use graph partitioning to minimise the need for communication
  - ORACLE implemented our query answering algorithm in their graph DB
- Improve query planning
  - Accurate join cardinality estimation crucial
  - Existing approaches quite rudimentary:
    - No formal foundations $\Rightarrow$ ad hoc
    - Only one-dimensional sampling
    - Predicate independence assumption quite crude
  - We are investigating an approach based on graph summarisation
    - Clear statistical interpretation of the estimates
- Exploit the theory of queries of bounded treewidth
  - Queries are often very large ($>$ 20 atoms), but of small treewidth
  - Preliminary experiments show great potential