Reasoning and Querying Web-scale Open Data based on DL-Lite$_{\mathcal{A}}$ in a Divide-and-Conquer Way

Zhenzhen Gu, Songmao Zhang and Cungen Cao

Institute of Computing Technology
Chinese Academy of Sciences
Beijing China

October 28, 2019
ISWC 2019 October 26-31, Aucland, New Zealand

Introduction

Introduction—Discussed problem

Positive sign:

More and more open datasets are described and published on the web based on Semantic Web standards, such as OWL and RDFs, *e.g. LOD*.

Caused Issues:

- The sheer size of these datasets ⇒
  performance bottlenecks for systems and tools that provide data managing and query answering services over the LOD datasets
- The growth of schema knowledge described by OWL ⇒
  realizing efficient reasoning and expressive query answering a challenging task for both data publishers and end users.

Addressed Probelm:

How to efficiently and powerfully consume the web-scale Open Data.

Introduction—Statistic of the BTC 2012 datasets

BTC 2012 Dataset: 1.1 billion RDF triples obtained by crawling from the Open Web.

Statistic Results:

- 99% are individual assertions: $A(a), P(a, b)$;
- 344,778 axioms, 339,519 (98%) — Captured by DL-Lite$_{\mathcal{A}}$:
  - $15,932 : A \sqsubseteq \exists R.B$
  - $4,625 : A \sqsubseteq \neg B$
  - $1,854 : P = S^-$
  - $614 : Fun(P)$
  - $167 : P = p^-$
  - *other forms* (RDFS)

Introduction—DL-Lite$_\mathcal{A}$

DL-Lite$_\mathcal{A}$: lightweight DL designed for Ontology Based Data Access

Reasoning Reduction: For a given DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$

$$\begin{cases} \mathcal{K} \text{ is satisfiable} \Leftrightarrow \cup_{Q \in \mathsf{Violates}(\mathcal{T})} \mathsf{Ans}(Q, (\emptyset, \mathcal{A})) = \emptyset; \\ \text{If } \mathcal{K} \text{ is satisfiable, then for each conjunctive query } Q: \\ \quad \mathsf{Ans}(Q, \mathcal{K}) = \cup_{Q' \in \mathsf{PerfectRef}(Q, \mathcal{T})} \mathsf{Ans}(Q', (\emptyset, \mathcal{A})) \end{cases}$$

$$\Downarrow$$

Outstanding characters:

- Low reasoning complexity
  - AC_0 data complexity — Satisfiability Checking
  - PTime KB complexity — Query Answering
- Separation between schema knowledge and data layer reasoning $\Rightarrow$ *make it possible to use highly optimized database management systems in practice for query answering*

Introduction—Our Proposal and the Bottleneck

Our Proposal:

Use the techniques of DL-Lite$_{\mathcal{A}}$ to consume the web-scale Open Data.

Bottleneck:

1. For satisfiability checking and conjunctive query answering, *polynomial sizes* of queries may need to be evaluated over the data layers (ABoxes) of the KBs, w.r.t. the size of the schema knowledge.

   Example: If a KB uses the DBpedia ontology as schema knowledge,

   - Satisfiability checking: $2,963,132$ queries;
   - If a query contains the class dbo:Person and property rdfs:label:
     more than $1,144 \times 1,391 = 1,591,304$ queries

2. For the KBs with *massive individual assertions*, even aided with highly optimized database management systems, evaluating a single query over the data layers may be very time-consuming.

Introduction—Our Solution

Divide-and-Conquer:

$$\begin{cases} \mathcal{K}, Q \xrightarrow{\text{KB and Query Prtition}} \mathcal{S} = \{\mathcal{K}_1, \cdots, \mathcal{K}_n\}, \mathcal{Q} = \{Q_1, \cdots, Q_m\} \\ \text{CheckSat.}(\mathcal{K}), \text{Ans}(Q, \mathcal{K}) \xrightarrow{\text{Reasoning Reduction}} \text{CheckSat.}(\mathcal{K}_i), \text{Ans}(Q_j, \mathcal{K}_i) \end{cases}$$

Performance Improvement:

1. Smaller sizes of schema knowledge and queries $\Rightarrow$ *smaller numbers of queries needed to be answered over the data layers*;
2. Smaller sizes of the data layers $\Rightarrow$ *the eventual queries be evaluated more efficiently*;
3. Independence among the subtasks $\Rightarrow$ *distribution and parallelization techniques to take effect to improve the overall performance*.

Challenges:

- How to realize KB and query partition;
- How to realize reasoning reduction in a sound and complete way.

Introduction—Our Solution

Principles:

KB chunks: local feature of satisfiability checking and simple-query answering

1. $\mathcal{K}$ *is satisfiable* $\leftrightarrow$ *each* $\mathcal{K}_i$ *is satisfiable*;
2. *for each simple-query* q, $\mathsf{Ans}(q, \mathcal{K}) = \cup_{i=1}^{n} \mathsf{Ans}(q, \mathcal{K}_i)$.

Query partitions :

Partition Q *into simple-queries* $Q_1 - Q_m$, $\mathsf{Ans}(Q_i, \mathcal{K}_j) \rightarrow \mathsf{Ans}(Q, \mathcal{K})$

Simple-queries:

A conjunctive query is called a simple-query if all the query atoms share a common individual or variable. *Example:*

$$Person(?x) \wedge birthDate(?x, ?y) \wedge member(?x, MIT) \rightarrow q(?x, ?y)$$
$$subject(?x, ?y) \wedge boarder(?y, ?z) \rightarrow q(?x, ?y, ?z)$$
$$director(?x, ?y) \wedge writer(?z, ?y) \wedge Film(?y) \rightarrow q(?x, ?y, ?z)$$

$$Person(?x) \wedge prize(?x, ?y) \wedge comment(?y, ?z) \rightarrow q(?x, ?y, ?z)$$

Introduction—Our Contribution

Contribution:

- Divide-and-Conquer
  - *DL-Lite$_\mathcal{A}$ KB partition*
  - *Conjunctive query partition and evaluation*;
  - *Optimization of answering query partitions over KB partitions*;
- Experiments on actual Open Data to validate the proposal and approach.

This is the first study on partitioning DL-Lite$_\mathcal{A}$ KBs and queries for the purpose of reasoning and querying large-scale datasets.

D&C: DL-Lite$_\mathcal{A}$ KB Partition

from both theoretical and $\Downarrow$ *practical point of views*

1. Definition of DL-Lite$_\mathcal{A}$ KB partitions and SCSQA-local partitions: $\mathcal{K} = (\mathcal{T}, \mathcal{A}), \mathcal{S} = \cup_{i=1}^{n}\{(\mathcal{T}_i, \mathcal{A}_i)\}$;
2. **Iff** conditions of detecting SCSQA-local partitions of KBs; $\mathcal{S}$ is a SCSQA-local partition if for each $a \in \mathsf{Ind}(\mathcal{A})$ there exists $\mathcal{A}_i$ such that $\mathcal{A}_a \subseteq \mathcal{A}_i$
3. Concrete ways of computing SCSQA-local partitions of KBs $O(|\mathcal{A}|)$, $O(|\mathcal{T}|^2)$

D&C: Conjunctive query partition and evaluation

from both theoretical and $\Downarrow$ *practical point of views*

1. Definition of query partitions and simple-query reducible queries:
   Query Q, Partition $\mathcal{Q} = \{q_1, \cdots, q_m\}$, $\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S})$;
2. **Iff** conditions of detecting simple-query reducible queries;
   $O(|Q|)$
3. Concrete ways of partitioning and answering simple-query reducible queries Q and non-simple-query reducible queries Q':
   $\mathsf{Ans}(Q, \mathcal{K}) = \mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S})$, $\mathsf{Ans}(Q', \mathcal{K}) = \cup_{i=1}^{k} \mathsf{Ans}(Q, \mathcal{Q}_i, \mathcal{S})$
   $O(|Q|), O(|Q'|^2)$

D&C: Optimization of evaluating query partitions over KB partitions

for improving $\Downarrow$ *overall performance*

Proposing a *special a sub-query value transfer* procedure to reduce the intermediate results as many and as early as possible.

$$\mathsf{Ans}(q,(\mathcal{T},\mathcal{A})) \xrightarrow{\xi_1,\cdots,\xi_k} \cup_{i=1}^{k}\mathsf{Ans}(q\xi_i,(\mathcal{T},\mathcal{A}))$$
$$\cup_{i=1}^{k}\mathsf{Ans}(q\xi_i,(\mathcal{T},\mathcal{A})) = \cup_{q'\in\mathsf{PerfectRef}(q,\mathcal{T})}\cup_{i=1}^{n}\mathsf{Ans}(q'\xi_i,(\emptyset,\mathcal{A}))$$

KB $\mathcal{K}$, partition $\mathcal{S}$ of $\mathcal{K}$, query Q, partition $\mathcal{Q}=\cup_{i=1}^{m}\{q_i\}$ of Q:

$$\mathsf{Ans}(Q,\mathcal{Q},\mathcal{S}) = \{\vec{x}[\vec{x}_1/\vec{u}_1,\cdots,\vec{x}_n/\vec{u}_n] | (\vec{u}_1,\cdots,\vec{u}_n)\in \Lambda_1\times\cdots\times\Lambda_n\}$$

where $\vec{x}=\mathsf{hd}(Q)$, $\vec{x}_i=\mathsf{hd}(q_i)$ and $\Lambda_i=\cup_{\mathcal{K}'\in\mathcal{S}}\mathsf{Ans}(q_i,\mathcal{K}')$, and for each $1\leq i,j\leq n$, $1\leq k\leq|\vec{u}_i|$ and $1\leq l\leq|\vec{u}_j|$, if $\vec{x}_i[k]=\vec{x}_j[l]$ then $\vec{u}_i[k]=\vec{u}_j[l]$.

Experiments:

$\Downarrow$

Validate the efficiency of the divide-and-conquer approach
Validate the rationality of our proposal of using DL-Lite$_{\mathcal{A}}$

Experiments—Datasets and Tools

Datasets:

- BTC 2012 dataset: *1.1 billion RDF triples*
- DBpedia_200 dataset extended with DBpedia ontology: *0.278 billion RDF triples*

Tools:

- Algorithms
  - Consistent → *satisfiability checking*
  - PerfectRef → *conjunctive query rewriting*
- Systems : MySQL → *storing ABoxes and evaluating queries*
- Techniques : Multithread → *parallelization*

Experiments—KB Construction and Partition

KB Construction:

$$\begin{cases} \text{BTC 2012} \Longrightarrow \mathcal{K}_{btc} = (\mathcal{T}_{btc}, \mathcal{A}_{btc}) \\ \qquad\qquad |\mathcal{T}_{btc}| = 339,519 \text{ and } |\mathcal{A}_{btc}| = 1,040,455,793 \\ \text{DBpedia} \Longrightarrow \mathcal{K}_{dbp} = (\mathcal{T}_{dbp}, \mathcal{A}_{dbp}) \\ \qquad\qquad |\mathcal{T}_{dbp}| = 233,227 \text{ and } |\mathcal{A}_{dbp}| = 275,710,447 \end{cases}$$

KB Partition:

$$\begin{cases} \mathcal{K}_{btc} \xrightarrow{\text{SCSQA-local partition}} \mathcal{S}_{btc} = \cup_{i=1}^{30}\{\mathcal{K}_i^{btc} = (\mathcal{T}_i^{btc}, \mathcal{A}_i^{btc})\} \\ \qquad\qquad |\mathcal{T}_i^{btc}| \leq \textbf{7354} \text{ and } |\mathcal{A}_i^{btc}| \leq \textbf{52.02} \text{ million} \\ \mathcal{K}_{dbp} \xrightarrow{\text{SCSQA-local partition}} \mathcal{S}_{dbp} = \cup_{i=1}^{14}\{\mathcal{K}_i^{dbp} = (\mathcal{T}_i^{dbp}, \mathcal{A}_i^{dbp})\} \\ \qquad\qquad |\mathcal{T}_i^{dbp}| \leq \textbf{95,469} \text{ and } |\mathcal{A}_i^{dbp}| \leq \textbf{28.57} \text{ million} \end{cases}$$

Experiments—Satisfiability Checking

Motivation:

Detecting the effects of KB partition on the performance of satisfiability checking.

	$\mathcal{K}_{dbp}$	$\mathcal{S}_{dbp}$
Time	6.44	**0.07**
Result	False	False

	$\mathcal{K}_{btc}$	$\mathcal{S}_{btc}$
Time	++	**0.15**
Result	++	False

⇓

KB	$\mathcal{K}_{dbp}$	$\mathcal{K}_i^{dbp}$
Query	2,963,132	$\leq$ **402,145**

KB	$\mathcal{K}_{btc}$	$\mathcal{K}_j^{btc}$
Query	++	$\leq$ **309,665**

⇓

Indication:

KB partition equipped with parallelization can improve the performance of satisfiability checking significantly.

Experiments—Query Answering

Motivation:

Detecting the effects of KB partition, query partition as well as sub-query value transfer on the performance of query answering.

Tested Queries and Measurements:

$$\text{Queries} \begin{cases} \texttt{BTC2012}: & SQ_1^{btc} - SQ_8^{btc} \text{ and } NQ_1^{btc} - NQ_6^{btc}; \\ \texttt{DBpedia}: & SQ_1^{dbp} - SQ_8^{dbp} \text{ and } NQ_1^{dbp} - NQ_6^{dbp}. \end{cases}$$

$$\text{Measurements} \begin{cases} \texttt{Time}: & \text{Total seconds used;} \\ \texttt{Ans}: & \text{Total certain answers obtained;} \\ \texttt{RCQ}: & \text{Total number of rewritten queries.} \end{cases}$$

Formulas of computing RCQ in different situations:

$$\texttt{RCQ} = \begin{cases} |\mathsf{PerfectRef}(\mathcal{T}, Q)| & \text{Without KB and query partition;} \\ \sum_{k=1}^{m} |\mathsf{PerfectRef}(\mathcal{T}, q_i)| & \text{Solely with query partition;} \\ \max_{i=1}^{n}\{|\mathsf{PerfectRef}(\mathcal{T}_i, Q)|\} & \text{Solely with KB partition;} \\ \max_{i=1}^{n}\{\sum_{k=1}^{m} |\mathsf{PerfectRef}(\mathcal{T}_i, q_k)|\} & \text{With both KB and query partition.} \end{cases}$$

Experiments—Simple-query answering

		SQ_1^{dbp}	SQ_2^{dbp}	SQ_3^{dbp}	SQ_4^{dbp}	SQ_5^{dbp}	SQ_6^{dbp}	SQ_7^{dbp}	SQ_8^{dbp}
RCQ	$\mathcal{K}_{dbp}$	1,145	1	1,145	1	503	1	18,289	124
	$\mathcal{S}_{dbp}$	**378**	1	**378**	1	**253**	1	**377**	**12**
Time	$\mathcal{K}_{dbp}$	54.66	16.02	–	14.30	101.34	–	–	–
	$\mathcal{S}_{dbp}$	**29.97**	**4.08**	**201.86**	**1.40**	**45.74**	**824.03**	**30.51**	**49.44**
Ans	$\mathcal{K}_{dbp}$	846,391	5	–	728	25	–	–	–
	$\mathcal{S}_{dbp}$	846,391	5	1	728	25	162	70	7

		SQ_1^{btc}	SQ_2^{btc}	SQ_3^{btc}	SQ_4^{btc}	SQ_5^{btc}	SQ_6^{btc}	SQ_7^{btc}	SQ_8^{btc}
RCQ	$\mathcal{K}_{btc}$	491	91	44,591	1,933	100,829	1,093	491	11,649
	$\mathcal{S}_{btc}$	**133**	**28**	**4,126**	**239**	**577**	**197**	**166**	**1,317**
Time	$\mathcal{K}_{btc}$	224.98	114.87	–	12.87	4.45	4.11	55.01	–
	$\mathcal{S}_{btc}$	**57.35**	**11.07**	**492.49**	**3.27**	**2.21**	**2.44**	**26.49**	**191.53**
Ans	$\mathcal{K}_{btc}$	13.6 M	98	–	15	3	72	162	–
	$\mathcal{S}_{btc}$	13.6 M	98	96	15	3	72	162	26

Results:

No matter a simple-query has a large or small number of rewritten queries over the original KBs, or it can be evaluated efficiently or inefficiently over the original KB, KB partitioning equipped with parallelization can make this query be answered in an extremely efficient way.

Experiments—Non-simple-query answering

pp (npp) – with (without) query partition, vt (nvt) – with (without) query value transfer

		SQ_3^{dbp}	SQ_6^{dbp}	NQ_1^{dbp}	NQ_2^{dbp}	NQ_3^{dbp}	NQ_4^{dbp}	NQ_5^{dbp}	NQ_6^{dbp}
	$\mathcal{K}_{dbp}^{npp}$	–	–	–	1.07	60.75	–	–	–
T	$\mathcal{K}_{dbp}^{pp+nvt}$	75.40	–	106.02	–	–	–	–	–
i	$\mathcal{K}_{dbp}^{pp+vt}$	51.99	122.81	103.79	1.16	61.21	–	–	–
m	$\mathcal{S}_{dbp}^{npp}$	201.86	824.03	%	%	%	%	%	%
e	$\mathcal{S}_{dbp}^{pp+nvt}$	38.80	–	**22.82**	–	–	686.11	208.89	–
	$\mathcal{S}_{dbp}^{pp+vt}$	**10.80**	**63.33**	31.61	**0.49**	**18.75**	**679.53**	**14.78**	**188.96**

		SQ_3^{btc}	SQ_8^{btc}	NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
	$\mathcal{K}_{btc}^{npp}$	–	–	11.91	3.03	–	–	–	–
T	$\mathcal{K}_{btc}^{pp+nvt}$	–	–	–	–	–	–	–	–
i	$\mathcal{K}_{btc}^{pp+vt}$	–	91.03	4.54	1.93	37.95	–	117.63	112.86
m	$\mathcal{S}_{btc}^{npp}$	492.49	191.53	%	%	%	%	%	%
e	$\mathcal{S}_{btc}^{pp+nvt}$	–	–	9.88	–	–	–	–	–
	$\mathcal{S}_{btc}^{pp+vt}$	**361.65**	**12.08**	**2.38**	**0.61**	**6.03**	**213.71**	**43.23**	**39.68**

Main Result:

When a conjunctive query cannot be evaluated over the original KB efficiently, query partitioning combined with KB partitioning and sub-query value transfer can make this query to be evaluated in a very efficient way.

Experiments—Non-simple-query answering

pp (npp) – with (without) query partition, vt (nvt) – with (without) query value transfer

		SQ_3^{dbp}	SQ_6^{dbp}	NQ_1^{dbp}	NQ_2^{dbp}	NQ_3^{dbp}	NQ_4^{dbp}	NQ_5^{dbp}	NQ_6^{dbp}
RCQ	$\mathcal{K}_{dbp}^{npp}$	1,145	**1**	**1**	4	**1**	4,237	29,732	–
	$\mathcal{K}_{dbp}^{pp+nvt}$	1,146	2	2	5	2	1,065	2,302	–
	$\mathcal{K}_{dbp}^{pp+vt}$	1,146	2	2	5	2	1,065	2,302	–
	$\mathcal{S}_{dbp}^{npp}$	**378**	**1**	%	%	%	%	%	%
	$\mathcal{S}_{dbp}^{pp+nvt}$	379	2	2	**2**	2	**126**	**385**	**12,958**
	$\mathcal{S}_{dbp}^{pp+vt}$	379	2	2	**2**	2	**126**	**385**	**12,958**

		SQ_3^{btc}	SQ_8^{btc}	NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
RCQ	$\mathcal{K}_{btc}^{npp}$	44,591	11,649	91	91	8,281	123,873	312,131	3,299,661
	$\mathcal{K}_{btc}^{pp+nvt}$	**189**	120	92	92	182	17,704	3,522	656
	$\mathcal{K}_{btc}^{pp+vt}$	**189**	120	92	92	182	17,704	3,522	656
	$\mathcal{S}_{btc}^{npp}$	4,126	1,317	%	%	%	%	%	%
	$\mathcal{S}_{btc}^{pp+nvt}$	191	**70**	**29**	**29**	**56**	**1,318**	**305**	**197**
	$\mathcal{S}_{btc}^{pp+vt}$	191	**70**	**29**	**29**	**56**	**1,318**	**305**	**197**

Reason:

Query partition can reduce the number of rewritten queries significantly, especially in the situation of KB partition.

Experiments—Non-simple-query answering

pp (npp) – with (without) query partition, vt (nvt) – with (without) query value transfer

		SQ_3^{dbp}	SQ_6^{dbp}	NQ_1^{dbp}	NQ_2^{dbp}	NQ_3^{dbp}	NQ_4^{dbp}	NQ_5^{dbp}	NQ_6^{dbp}	
Time	$\mathcal{K}_{dbp}^{npp}$	–	–	–	–	1.07	60.75	–	–	–
	$\mathcal{K}_{dbp}^{pp+nvt}$	75.40	–	106.02	–	–	–	–	–	–
	$\mathcal{K}_{dbp}^{pp+vt}$	51.99	122.81	103.79	1.16	61.21	–	–	–	
	$\mathcal{S}_{dbp}^{npp}$	201.86	824.03	%	%	%	%	%	%	
	$\mathcal{S}_{dbp}^{pp+nvt}$	38.80	–	**22.82**	–	–	686.11	208.89	–	
	$\mathcal{S}_{dbp}^{pp+vt}$	**10.80**	**63.33**	31.61	**0.49**	**18.75**	**679.53**	**14.78**	**188.96**	

		SQ_3^{btc}	SQ_8^{btc}	NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
Time	$\mathcal{K}_{btc}^{npp}$	–	–	11.91	3.03	–	–	–	–
	$\mathcal{K}_{btc}^{pp+nvt}$	–	–	–	–	–	–	–	–
	$\mathcal{K}_{btc}^{pp+vt}$	–	91.03	4.54	1.93	37.95	–	117.63	112.86
	$\mathcal{S}_{btc}^{npp}$	492.49	191.53	%	%	%	%	%	%
	$\mathcal{S}_{btc}^{pp+nvt}$	–	–	9.88	–	–	–	–	–
	$\mathcal{S}_{btc}^{pp+vt}$	**361.65**	**12.08**	**2.38**	**0.61**	**6.03**	**213.71**	**43.23**	**39.68**

Reason:

In the situation of query partition, sub-query value transfer can make the finally rewritten queries be evaluated much more quickly.

Experiments—Non-simple-query answering

pp (npp) – with (without) query partition, vt (nvt) – with (without) query value transfer

		SQ_3^{dbp}	SQ_6^{dbp}	NQ_1^{dbp}	NQ_2^{dbp}	NQ_3^{dbp}	NQ_4^{dbp}	NQ_5^{dbp}	NQ_6^{dbp}
	$\mathcal{K}_{dbp}^{npp}$	–	–	–	1.07	60.75	–	–	–
T	$\mathcal{K}_{dbp}^{pp+nvt}$	75.40	–	106.02	–	–	–	–	–
i	$\mathcal{K}_{dbp}^{pp+vt}$	51.99	122.81	103.79	1.16	61.21	–	–	–
m	$\mathcal{S}_{dbp}^{npp}$	201.86	824.03	%	%	%	%	%	%
e	$\mathcal{S}_{dbp}^{pp+nvt}$	38.80	–	22.82	–	–	686.11	208.89	–
	$\mathcal{S}_{dbp}^{pp+vt}$	10.80	63.33	31.61	**0.49**	**18.75**	679.53	14.78	188.96

		SQ_3^{btc}	SQ_8^{btc}	NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
	$\mathcal{K}_{btc}^{npp}$	–	–	11.91	3.03	–	–	–	–
T	$\mathcal{K}_{btc}^{pp+nvt}$	–	–	–	–	–	–	–	–
i	$\mathcal{K}_{btc}^{pp+vt}$	–	91.03	4.54	1.93	37.95	–	117.63	112.86
m	$\mathcal{S}_{btc}^{npp}$	492.49	191.53	%	%	%	%	%	%
e	$\mathcal{S}_{btc}^{pp+nvt}$	–	–	9.88	–	–	–	–	–
	$\mathcal{S}_{btc}^{pp+vt}$	361.65	12.08	**2.38**	**0.61**	6.03	213.71	43.23	39.68

Weakness:

For the queries already efficiently answerable over the original KBs, our approach may not future improve the performance.

Experiments—Comparison with related systems

Motivation:

Validating the rationality of the proposal of using DL-Lite$_{\mathcal{A}}$ as well as the significance of KB and query partition.

Compared systems:

Jena TDB (version 2.10.1) and Virtuoso (version 7.2.4.2): Centralized triple stores, adopting the triple table model to manage RDF graphs and supporting lightweight RDFS reasoning.

Experiments—Comparison with related systems

Vir *(TDB)–Evaluating queries with Virtuoso (Jena TDB) without reasoning*

		SQ_1^{btc}	SQ_2^{btc}	SQ_3^{btc}	SQ_4^{btc}	SQ_5^{btc}	SQ_6^{btc}	SQ_7^{btc}	SQ_8^{btc}
T	Our	57.35	11.07	361.65	3.27	2.21	2.44	26.49	12.08
i	Vir	127.41	1.27	1.46	**2.65**	**0.22**	**1.35**	**0.42**	**1.46**
m	TDB	305.74	**0.37**	**0.37**	4.63	0.30	5.54	44.32	3.82
e	ViR	–	30.46	–	28.87	29.99	30.77	–	45.29
	TDR	–	25.91	–	19.52	21.42	24.28	–	65.80

		NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
T	Our	2.38	0.61	6.03	213.71	43.23	39.68
i	Vir	**0.21**	**0.28**	**0.25**	**0.41**	2.41	**2.33**
m	TDB	–	13.01	0.42	1.50	**0.21**	5.90
e	ViR	29.87	29.44	29.81	*72.99*	–	–
	TDR	–	105.31	18.60	157.94	–	–

Aspect one:

Even with KB and query partitioning, our approach is not as efficient as the two RDF stores which evaluate queries directly without taking reasoning into consideration.

Experiments—Comparison with related systems

ViR (TDR)–*Evaluating queries with Virtuoso (Jena TDB) with RDFS reasoning*

		SQ_1^{btc}	SQ_2^{btc}	SQ_3^{btc}	SQ_4^{btc}	SQ_5^{btc}	SQ_6^{btc}	SQ_7^{btc}	SQ_8^{btc}
T	Our	57.35	11.07	361.65	3.27	2.21	2.44	26.49	12.08
i	ViR	–	30.46	–	28.87	29.99	30.77	–	45.29
m	TDR	–	25.91	–	19.52	21.42	24.28	–	65.80
e	Vir	127.41	1.27	1.46	**2.65**	**0.22**	**1.35**	**0.42**	**1.46**
	TDB	305.74	**0.37**	**0.37**	4.63	0.30	5.54	44.32	3.82

		NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
T	Our	2.38	0.61	6.03	213.71	43.23	39.68
i	ViR	29.87	29.44	29.81	*72.99*	–	–
m	TDR	–	105.31	18.60	157.94	–	–
e	Vir	**0.21**	**0.28**	**0.25**	**0.41**	2.41	**2.33**
	TDB	–	13.01	0.42	1.50	**0.21**	5.90

Aspect two:

When reasoning is considered, the performance of the two triple stores drops dramatically, and they do not perform as well as our approach. $\Rightarrow$ *When reasoning is taken into account, KB and query partitioning has the potential to improve the performance of query answering significantly.*

Experiments—Comparison with related systems

		SQ_1^{btc}	SQ_2^{btc}	SQ_3^{btc}	SQ_4^{btc}	SQ_5^{btc}	SQ_6^{btc}	SQ_7^{btc}	SQ_8^{btc}
	Our	**13M**	**98**	**96**	**15**	3	**72**	**162**	26
A	Vir	12M	7	5	0	3	51	138	26
n	TDB	12M	7	5	0	3	51	138	26
s	ViR	–	**98**	–	**15**	3	**72**	–	26
	TDR	–	**98**	–	**15**	3	**72**	–	26

		NQ_1^{btc}	NQ_2^{btc}	NQ_3^{btc}	NQ_4^{btc}	NQ_5^{btc}	NQ_6^{btc}
	Our	**843**	**36**	**38**	25	**2,092**	**1,685**
A	Vir	1	4	34	25	0	1,187
n	TDB	1	4	34	25	0	1,187
s	ViR	**843**	**36**	**38**	25	–	–
	TDR	–	**36**	**38**	25	–	–

Aspect three:

Query answering without considering reasoning can miss many certain answers.

Comparison with related works

Comparison with related works

For Linked Data query answering:
We consider Linked Data as DL-Lite$_\mathcal{A}$ KBs.

For KB modularity and partitioning:
We focus on DL-Lite$_\mathcal{A}$, and discuss satisfiability checking and query answering rather than axiom and assertion entailment checking.

For query rewriting optimization:
Our approach features query rewriting optimization from the perspective of query partitioning.

Conclusion and Future Work

Conclusion

1. To capture the abundant schema knowledge, we propose to use the techniques of DL-Lite$_{\mathcal{A}}$ to consume the web-scale open data;
2. For web-scalability and efficiency, we provide a divide-and-conquer reasoning and query answering approach for DL-Lite$_{\mathcal{A}}$ as well as the corresponding optimization strategy;
3. We conduct experiments to validate the rationality and efficiency of our proposal and the provided method.

Future Work

This is the first step towards using DL-Lite$_\mathcal{A}$ techniques to consume the web-scale Open Data.

1. Analyzing the impact of the number and size of the sub-KBs in a KB partition on the performance of satisfiability checking and query answering;
2. Discussing partitioning DL-Lite$_\mathcal{A}$ KBs and conjunctive queries based on other principles;
3. Applying the main idea of the overall approach to other DL languages.

Thank you very much!

The technique details can be found in the paper:

Z, Gu., S, Zhang., and C, Cao. Reasoning and querying web-scale open data based on DL-LiteA in a divide-and-conquer way.
J. Web Semant. 55: 122-144 (2019)

DL-Lite$_{\mathcal{A}}$ KB Partition:

from both theoretical and $\Downarrow$ *practical point of views*

Definition of DL-Lite$_{\mathcal{A}}$ KB partitions
Iff conditions of detecting the desired KB partitions
Concrete ways of computing the desired KB partitions

KB partition—Formalization of DL-Lite$_\mathcal{A}$ KB partitions

Definition: (*DL-Lite$_\mathcal{A}$ KB partitions*)

For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, the set $\mathcal{S}$ of DL-Lite$_\mathcal{A}$ KBs:

$$\mathcal{S} = \{(\mathcal{T}_1, \mathcal{A}_1), \cdots, (\mathcal{T}_n, \mathcal{A}_n)\}$$

is called a partition of $\mathcal{K}$ if $\mathcal{A} = \cup_{i=1}^{n} \mathcal{A}_i$ and $\mathcal{T}_i = \mathcal{T}|_{\mathcal{A}_i}$ for $1 \leq i \leq n$.

$\mathcal{S}$ is called a SCSQA-local partition of $\mathcal{K}$ iff (1) $\mathcal{K}$ is satisfiable iff each KB in $\mathcal{S}$ are satisfiable; and (2) if $\mathcal{K}$ is satisfiable then for each simple-query Q, $\mathsf{Ans}(Q, \mathcal{K}) = \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{Ans}(Q, \mathcal{K}')$.

Lemma: (*Correctness of computing smaller TBoxes for smaller ABoxes*)

1. $(\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')$ is satisfiable iff $(\mathcal{T}, \mathcal{A}')$ is satisfiable;
2. $\mathsf{Ans}(Q, (\mathcal{T}|_{\mathcal{A}'}, \mathcal{A}')) = \mathsf{Ans}(Q, (\mathcal{T}, \mathcal{A}'))$ for a conjunctive query Q.

KB partition— Conditions of detecting desired partitions

Basic Observation:

In DL-Lite$_\mathcal{A}$, both satisfiability checking and simple-query answering can be reduced to evaluating simple-queries over the ABoxes of KBs.

Lemma: For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$:

$$\begin{cases} \text{there exists a set } \mathcal{Q} \text{ of simple-queries} \Rightarrow \\ \mathcal{K} \text{ is satisfiable iff } \cup_{Q \in \mathcal{Q}} \mathsf{Ans}(Q, (\emptyset, \mathcal{A})) = \emptyset \\ \text{if } \mathcal{K} \text{ is satisfiable then for each simple-query } Q, \text{ there exists} \Rightarrow \\ \text{a set } \mathcal{Q} \text{ of simple-queries} \mathsf{Ans}(Q, \mathcal{K}) = \cup_{Q' \in \mathcal{Q}} \mathsf{Ans}(Q', (\emptyset, \mathcal{A})) \end{cases}$$

KB partition— Conditions of detecting desired partitions

Proposition: (*Sufficient detecting condition*)

For a DL-Lite$_{\mathcal{A}}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ and partition $\mathcal{S}$ of $\mathcal{K}$, if for each $U \subseteq \mathcal{A}$ such that all the assertions in U share a common individual, there exists $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that $U \subseteq \mathcal{A}'$, then $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$.

Theorem: (*Sufficient and necessary detecting condition*)

For a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ without redundant individual assertions, a partition $\mathcal{S}$ is a SCSQA-local partition of $\mathcal{K}$ iff for each subset U $\subseteq \mathcal{A}$ satisfying that all the assertions in U share a common individual, there exists $(\mathcal{T}', \mathcal{A}') \in \mathcal{S}$ such that U $\subseteq \mathcal{A}'$.

Theorem: (*Sufficient and necessary detecting condition*)

A partition $\mathcal{S}$ of a KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ is a SCSQA-local partition of $\mathcal{K}$, ff for each individual assertion set U satisfying that all the assertions in U share a common individual and $\mathcal{K} \models \alpha$ for each $\alpha \in$ U, there exists $\mathcal{K}' \in \mathcal{S}$ such that $\mathcal{K}' \models \alpha$ for each $\alpha \in$ U.

KB partition—Computing SCSQA-local partitions

Procedure: (*Computing SCSQA-local partitions of DL-Lite$_\mathcal{A}$ KBs:*)

A SCSQA-local partition $\mathcal{S}$ of a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ can be obtained the two steps. Concretely:

1. Compute a partition $\mathcal{A}_1, \cdots, \mathcal{A}_n$ of $\mathcal{A}$ such that for each $a \in \mathsf{Ind}(\mathcal{A})$, there exists $\mathcal{A}_i$ such that $\mathcal{A}_{\{a\}} \subseteq \mathcal{A}_i$;
2. For each $1 \leq i \leq n$, compute the sub-TBox $\mathcal{T}|_{\mathcal{A}_i}$ for $\mathcal{A}_i$. Let $\mathcal{S} = \{(\mathcal{T}|_{\mathcal{A}_1}, \mathcal{A}_1), \cdots, (\mathcal{T}|_{\mathcal{A}_n}, \mathcal{A}_n)\}$.

Theorem: (*Complexity of computing SCSQA-local partitions*)

For a DL-Lite$_\mathcal{A}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, a SCSQA-local partition of $\mathcal{K}$ can be computed in $O(|\mathcal{T}|^2)$ w.r.t. $|\mathcal{T}|$ and in $O(|\mathcal{A}|)$ w.r.t. $|\mathcal{A}|$.

Low computational complexity makes the proposed approach appealing for partitioning a large-scale DL-Lite$_\mathcal{A}$ KB into smaller, independent sub-KBs which have the local feature of SC and SQA.

Conjunctive query partition and evaluation:

from both theoretical and ⇓ *practical point of views*

Definition of conjunctive query partitions
Iff conditions of detecting simple-query reducible conjunctive queries
Concrete ways of partitioning and answering conjunctive queries

Query Partition—Formalization of query partitions

Definition: (*Conjunctive query partitions*)

For a conjunctive query Q, the set $\mathcal{Q}=\{q_1, \cdots, q_n\}$ of conjunctive queries is called a partition of Q if:

1. $\mathsf{bd}(Q) = \cup_{i=1}^{n} \mathsf{bd}(q_i)$, and $\mathsf{bd}(q_i) \cap \mathsf{bd}(q_j) = \emptyset$ when $i \neq j$; and
2. $\mathsf{hd}(q_i) = \mathsf{bdV}(q_i) \cap (\mathsf{hd}(q) \cup \cup_{j=1, j\neq i}^{n} \mathsf{bdV}(q_j)$

$\mathcal{Q}$ is a **simple-query partition** of Q iff each query in $\mathcal{Q}$ is a simple-query.

Query Partition—Formalization of query partitions

Formula: (*Merging certain answers of sub-queries*)

For a partition $\mathcal{Q} = \{q_1, \cdots, q_n\}$ of a conjunctive query Q and a partition $\mathcal{S}$ of a DL-Lite$_\mathcal{A}$ KB, we define:

$$\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S}) = \{\vec{x}[\vec{x}_1/\vec{u}_1, \cdots, \vec{x}_n/\vec{u}_n] | (\vec{u}_1, \cdots, \vec{u}_n) \in \Lambda_1 \times \cdots \times \Lambda_n\}$$

where $\vec{x} = \mathsf{hd}(Q)$, for each $1 \leq i \leq n$, $\vec{x}_i = \mathsf{hd}(q_i)$ and $\Lambda_i = \cup_{\mathcal{K}' \in \mathcal{S}} \mathsf{Ans}(q_i, \mathcal{K}')$, and for each $1 \leq i, j \leq n$, $1 \leq k \leq |\vec{u}_i|$ and $1 \leq l \leq |\vec{u}_j|$, if $\vec{x}_i[k] = \vec{x}_j[l]$ then $\vec{u}_i[k] = \vec{u}_j[l]$.

Lemma: (*Correctness of the merging procedure*)

For a conjunctive query Q and partition $\mathcal{Q}$ of Q, then for each conjunctive query Q and partition $\mathcal{Q}$ of Q, $\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S}) \subseteq \mathsf{Ans}(Q, \mathcal{K})$ holds.

Query Partition—The desired way of answering queries

Desired way:

$$\begin{cases} \text{For a conjunctive query } Q, \text{compute a simple-query partition } \mathcal{Q} \text{ of } Q; \\ \text{For a SCSQA-local partition } \mathcal{S} \text{ of a DL-Lite}_{\mathcal{A}} \text{ KB } \mathcal{K}, \text{ compute } \mathsf{Ans}(\mathcal{Q}, \mathcal{S}) \\ \text{to obtain } \mathsf{Ans}(Q, \mathcal{K}) \end{cases}$$

Simple-query reducible queries: The conjunctive queries can be answered in the desired way.

Query Partition—Detecting simple-query reducible queries

Example: (*Not all queries can be answered in the desired way*)

Considered the following conjunctive query Q:

$$Q\colon\ A(?x) \wedge P(?x,?y) \wedge B(?y) \rightarrow q()$$

$$\begin{cases} \mathcal{Q}_1 = \{A(?x) \wedge P(?x,?y) \rightarrow q(?y),\ \ B(?y) \rightarrow q(?y)\} \\ \mathcal{Q}_2 = \{A(?x) \rightarrow q(?x),\ \ P(?x,?y) \wedge B(?y) \rightarrow q(?x)\} \\ \mathcal{Q}_3 = \{A(?x) \rightarrow q(?x),\ \ P(?x,?y) \rightarrow q(?x,?y),\ \ B(?y) \rightarrow q(?y)\} \end{cases}$$

Consider the DL-Lite$_{\mathcal{A}}$ KB KB $\mathcal{K}$:

$$\mathcal{K} = (\{C \sqsubseteq \exists S,\ \exists S^- \sqsubseteq A,\ A \sqsubseteq \exists P,\ \exists P^- \sqsubseteq B\},\ \{C(a)\})$$

Ans$(Q, \mathcal{Q}_i, \{\mathcal{K}\}) = \emptyset$ for $1 \leq i \leq 3$. However, Ans$(Q, \mathcal{K}) = \{()\}$.

Query Partition—Detecting simple-query reducible queries

Motivation:

Answering as many conjunctive queries in the desired way as possible.

Theorem: (*Iff conditions of detecting simple-query reducible queries*)

A conjunctive query Q is **simple-query reducible iff** there do not exist three query atoms $P(?x, ?y)$, α and α' in Q such that $?x$ and $?y$ are non-distinguished variables of Q, α contains $?x$ but not $?y$, and α' contains $?y$ but not $?x$.

Theorem: (*Complexity of detecting simple-query reducible queries*)

For a conjunctive query Q, the complexity of deciding whether Q is simple-query reducible is $O(|Q|)$.

Query Partition—Answering queries through query partition

For simple-query reducible queries Q:

- Identifying the simple-query partitions $\mathcal{Q}$ that can capture all the certain answers of the original queries Q, i.e.,
  $\mathsf{Ans}(\mathcal{Q}, \mathcal{S}) = \mathsf{Ans}(Q, \mathcal{K})$ w.r.t. a KB $\mathcal{K}$ and its SCSQA-local partition $\mathcal{S}$.
- The concrete ways of computing such query partitions $\mathcal{Q}$.

For non-simple-query reducible queries Q:

- Identifying the partitions $\mathcal{Q}_1, \cdots, \mathcal{Q}_n$ that can capture all the certain answers of the original queries Q, i.e.,
  $\mathsf{Ans}(Q, \mathcal{K}) = \cup_{i=1}^{n} \mathsf{Ans}(\mathcal{Q}_i, \mathcal{S})$ w.r.t. a KB $\mathcal{K}$ and its SCSQA-local partition $\mathcal{S}$.
- The concrete ways of computing such query partitions $\mathcal{Q}$.

Optimization of evaluating query partitions over KB partitions

Optimization—Motivation

Problem:

A large amount of intermediate results generated by answering partitioned sub-queries may slow down the overall query answering procedure.

$$\text{Ans}(Q, \mathcal{Q}, \mathcal{S}) = \{\vec{x}[\vec{x}_1/\vec{u}_1, \cdots, \vec{x}_n/\vec{u}_n] | (\vec{u}_1, \cdots, \vec{u}_n) \in \Lambda_1 \times \cdots \times \Lambda_n\}$$

Motivation:

Provide strategies to **reduce the intermediate results** as many and as early as possible to improve the overall query answering procedure.

Optimization—Basic idea

Basic Idea: Value transfer

When answering a sub-query in a partition of a conjunctive query:

we will transfer the corresponding values obtained from the certain answers of the sub-queries that have already been evaluated to this sub-query before it is evaluated.

$$\Downarrow \; Advantage$$

Avoid computing the certain answers of this sub-query which do not coincide with the previous sub-queries.

Optimization—Basic idea

Example: (*Value transfer procedure*)

Let $\mathcal{S}$ be a partition of a DL-Lite$_\mathcal{A}$ KB. Consider the following conjunctive query Q and its partition $\mathcal{Q}$:

$$\begin{cases} Q: \ R(a,?x) \wedge P(?x,?y) \wedge S(?y,?z) \rightarrow q(?x,?y,?z) \\ \mathcal{Q} = \begin{cases} q_1 : R(a,?x) \wedge P(?x,?y) \rightarrow q(?x,?y), \\ q_2 : S(?y,?z) \rightarrow q(?y,?z) \end{cases} \end{cases}$$

After evaluating q_1 over the KBs in $\mathcal{S}$, suppose we obtain the answer set $\Lambda = \cup_{i=1}^{3}\{(a_i, b_i)\}$. If we transfer the bindings of $?y$ to q_2 before it is evaluated, i.e., replacing evaluating q_2 with answering these three queries:

$$\begin{cases} q_2^1 : S(b_1,?z) \rightarrow q(b_1,?z) \\ q_2^2 : S(b_2,?z) \rightarrow q(b_2,?z) \\ q_2^3 : S(b_3,?z) \rightarrow q(b_3,?z) \end{cases}$$

then we can avoid computing the certain answers of q_2 that do not bind $?y$ to b_1, b_2 or b_3.

Optimization—Formalization

Proposition: (*Formalization and correctness of value transfer*)

For partition $\mathcal{Q} = \{q_1, \cdots, q_n\}$ of a conjunctive query Q and partition $\mathcal{S}$ of a satisfiable DL-Lite$_\mathcal{A}$ KB, the following equation holds:

$$\mathsf{Ans}(Q, \mathcal{Q}, \mathcal{S}) = \{\vec{x}[\vec{x}_1/\vec{u}_1, \cdots, \vec{x}_n/\vec{u}_n] | (\vec{u}_1, \cdots, \vec{u}_n) \in \Lambda_1|_{\mathcal{B}_1} \times \cdots \times \Lambda_n|_{\mathcal{B}_n}\}$$

where $\vec{x} = \mathsf{hd}(Q)$, $\vec{x}_i = \mathsf{hd}(q_i)$ for $1 \leq i \leq n$, and

$$\mathcal{B}_k = \begin{cases} \emptyset & \text{if } k = 1; \\ \{[\vec{x}_1/\vec{u}_1, \cdots, \vec{x}_{k-1}/\vec{u}_{k-1}]|_{\vec{x}_k}| & \\ \qquad (\vec{u}_1, \cdots, \vec{u}_{k-1}) \in \Lambda_1|_{\mathcal{B}_1} \times \cdots \times \Lambda_{k-1}|_{\mathcal{B}_{k-1}}\} & \text{if } k > 1. \end{cases}$$

$$\Lambda_k|_{\mathcal{B}_k} = \begin{cases} \cup_{\mathcal{K} \in \mathcal{S}} \mathsf{Ans}(q_1, \mathcal{K}) & \text{if } k = 1; \\ \cup_{\mathcal{K} \in \mathcal{S}} \cup_{\xi \in \mathcal{B}_k} \mathsf{Ans}(q_k\xi, \mathcal{K}) & \text{if } k > 1. \end{cases}$$

Optimization—Optimized value transfer

Basic observation:

The transfered values do not affect the query rewriting procedure.

Formalized conclusions:

Sub-query q, DL-Lite$_{\mathcal{A}}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, transferred variable binding ξ:

$$\mathsf{PerfectRef}(q\xi, \mathcal{T}) = \cup_{\xi \in \mathcal{B}} \{q'\xi | q' \in \mathsf{PerfectRef}(q, \mathcal{T})\}$$

Thus, for a set $\mathcal{B} = \cup_{i=1}^{n} \{\xi_i\}$ of transferred variable bindings of q:

$$\begin{cases} \cup_{i=1}^{n} \mathsf{Ans}(q\xi_i, \mathcal{K}) = \cup_{i=1}^{n} \cup_{q' \in \mathsf{PerfectRef}(q\xi_i, \mathcal{T})} \mathsf{Ans}(q', (\emptyset, \mathcal{A})) \\ \Downarrow \\ \cup_{i=1}^{n} \mathsf{Ans}(q\xi_i, \mathcal{K}) = \cup_{q' \in \mathsf{PerfectRef}(q, \mathcal{T})} \cup_{i=1}^{n} \mathsf{Ans}(q'\xi_i, (\emptyset, \mathcal{A})) \end{cases}$$

So just one time query rewriting procedure is required rather than n times.

Optimization—Optimized value transfer

Example: (*Optimized value transfer*)

Query q, KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, transferred variable bindings $\cup_{i=1}^{3}\{\xi_i : ?y \to b_i\}$:

$$\cup_{i=1}^{3}\mathsf{Ans}(q\xi_i, \mathcal{K})$$

$$\Downarrow$$

$$\mathsf{PerfectRef}(q, \mathcal{T}) =$$

$$\{q_1 : S_1(?y, ?z) \to q(?y, ?z), \quad q_2 : S_2(?z, ?y) \to q(?y, ?z)\}$$

$$\Downarrow \qquad\qquad \Downarrow$$

$$\begin{array}{ll} q_1^1 : S_1(b_1, ?z) \to q(b_1, ?z), & q_2^1 : S_2(?z, b_1) \to q(b_1, ?z) \\ q_1^2 : S_1(b_2, ?z) \to q(b_1, ?z), & q_2^2 : S_2(?z, b_2) \to q(b_2, ?z) \\ q_1^3 : S_1(b_3, ?z) \to q(b_1, ?z), & q_2^3 : S_2(?z, b_3) \to q(b_3, ?z) \end{array}$$

$$\Downarrow$$

$$\cup_{i=1}^{3}\mathsf{Ans}(q_1^i, (\emptyset, \mathcal{A})) \cup \cup_{i=1}^{3}\mathsf{Ans}(q_2^i, (\emptyset, \mathcal{A}))$$