Aggregate Queries over Ontologies

D. Calvanese, E. Kharlamov, W. Nutt, C. Thorne

KRDB Centre

Free University of Bozen-Bolzano

Via della Mostra 4

39100 - Italy

The Problem

Ontology-based data access systems (OBDASs) aim at accessing (possibly from the web) large-size repositories of heterogenous data [Calvanese et al. 2005]

The Problem

Ontology-based data access systems (OBDASs) aim at accessing (possibly from the web) large-size repositories of heterogenous data [Calvanese et al. 2005]

In such scenario, ontologies provide a global **conceptual model** of all those potentially incomplete sources

The Problem

Ontology-based data access systems (OBDASs) aim at accessing (possibly from the web) large-size repositories of heterogenous data [Calvanese et al. 2005]

In such scenario, ontologies provide a global **conceptual model** of all those potentially incomplete sources

Querying thus takes place under the **open world assumption** (OWA)

The Problem

Ontology-based data access systems (OBDASs) aim at accessing (possibly from the web) large-size repositories of heterogenous data [Calvanese et al. 2005]

In such scenario, ontologies provide a global **conceptual model** of all those potentially incomplete sources

Querying thus takes place under the **open world assumption** (OWA)

Ontologies are expressed in a suitable ontology language (e.g. OWL) or in DLs

The Problem

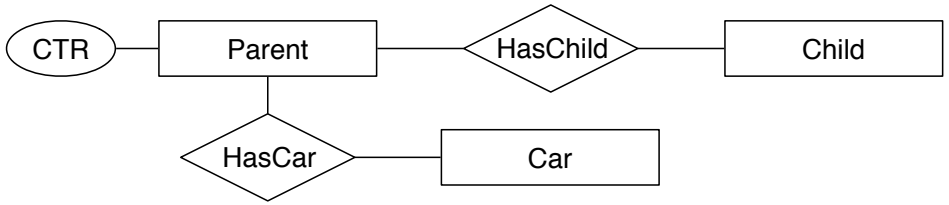
Ontology-based data access systems (OBDASs) aim at accessing (possibly from the web) large-size repositories of heterogenous data [Calvanese et al. 2005]

In such scenario, ontologies provide a global **conceptual model** of all those potentially incomplete sources

Querying thus takes place under the **open world assumption** (OWA)

Ontologies are expressed in a suitable ontology language (e.g. OWL) or in DLs

Consider the following `Family` ontology (expressed as an ER-diagram):



The Problem

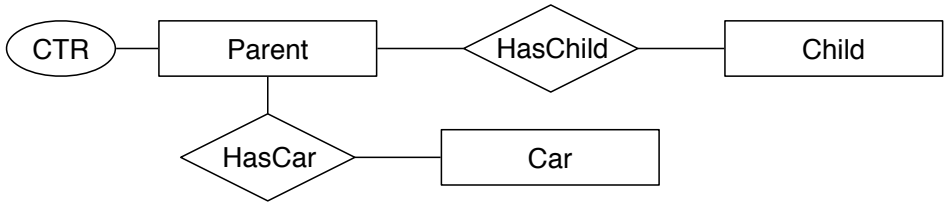
Ontology-based data access systems (OBDASs) aim at accessing (possibly from the web) large-size repositories of heterogenous data [Calvanese et al. 2005]

In such scenario, ontologies provide a global **conceptual model** of all those potentially incomplete sources

Querying thus takes place under the **open world assumption** (OWA)

Ontologies are expressed in a suitable ontology language (e.g. OWL) or in DLs

Consider the following `Family` ontology (expressed as an ER-diagram):



`Family` models (partly) how families receive in Italy subsidies for their children, whose data can come from arbitrary sources

The Problem

Among the meaningful queries we would like to to ask to `Family` are **aggregate queries** like:

The Problem

Among the meaningful queries we would like to to ask to `Family` are **aggregate queries** like:

(a) (**max**) What is the maximum per child tax reduction explicitly recorded in the data?

The Problem

Among the meaningful queries we would like to to ask to `Family` are **aggregate queries** like:

(a) (**max**) What is the maximum per child tax reduction explicitly recorded in the data?

(b) (**countd**) How many parents own cars?

The Problem

Among the meaningful queries we would like to to ask to `Family` are **aggregate queries** like:

(a) (**max**) What is the maximum per child tax reduction explicitly recorded in the data?

(b) (**countd**) How many parents own cars?

(c) (**sum**) What is the total tax reduction of each parent with children?

The Problem

Among the meaningful queries we would like to to ask to `Family` are **aggregate queries** like:

(a) (**max**) What is the maximum per child tax reduction explicitly recorded in the data?

(b) (**countd**) How many parents own cars?

(c) (**sum**) What is the total tax reduction of each parent with children?

Some research has been done for data integration and data exchange systems [Afrati & Kolaitis 2008, Arenas et al. 2003, Lechtenboerger et al. 2002]

The Problem

Among the meaningful queries we would like to to ask to `Family` are **aggregate queries** like:

(a) (**max**) What is the maximum per child tax reduction explicitly recorded in the data?

(b) (**countd**) How many parents own cars?

(c) (**sum**) What is the total tax reduction of each parent with children?

Some research has been done for data integration and data exchange systems [Afrati & Kolaitis 2008, Arenas et al. 2003, Lechtenboerger et al. 2002]

Less is known, however, about DL or OWL ontologies

Outline

Outline

1. Aggregate queries and DBs

Outline

1. Aggregate queries and DBs
2. Aggregate queries and ODBASs

Outline

1. Aggregate queries and DBs
2. Aggregate queries and ODBASs
3. Epistemic aggregate queries

Outline

1. Aggregate queries and DBs
2. Aggregate queries and ODBASs
3. Epistemic aggregate queries
3. Epistemic certain answers and their computation

Outline

1. Aggregate queries and DBs
2. Aggregate queries and ODBASs
3. Epistemic aggregate queries
3. Epistemic certain answers and their computation
4. Basic properties

Outline

1. Aggregate queries and DBs
2. Aggregate queries and ODBASs
3. Epistemic aggregate queries
3. Epistemic certain answers and their computation
4. Basic properties
5. Conclusions and further work

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema **R** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema $\mathbf{R}$ is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

– $q \notin \mathbf{R}$ is called a **head relation**

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema **R** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

– $q \notin \mathbf{R}$ is called a **head relation**

– α stands for an arbitrary SQL aggregation function

min, **max**, **count**, **countd**, **sum**, **avg**

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema **R** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

– $q \notin \mathbf{R}$ is called a **head relation**

– α stands for an arbitrary SQL aggregation function

min, **max**, **count**, **countd**, **sum**, **avg**

– ϕ is a list of relational atoms over **R** called **condition**

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema **R** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

– $q \notin \mathbf{R}$ is called a **head relation**

– α stands for an arbitrary SQL aggregation function

min, **max**, **count**, **countd**, **sum**, **avg**

– ϕ is a list of relational atoms over **R** called **condition**

– $[\psi]$ is a **nested condition**

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema **R** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

– $q \notin \mathbf{R}$ is called a **head relation**

– α stands for an arbitrary SQL aggregation function

min, **max**, **count**, **countd**, **sum**, **avg**

– ϕ is a list of relational atoms over **R** called **condition**

– $[\psi]$ is a **nested condition**

– $\vec{x}$ is a sequence of **grouping variables** and $Var(q) \subseteq Var(\phi, \psi)$

Conditional Aggregate Queries (CAQs)

DEF. A **conditional aggregate query** over a schema **R** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$$

– $q \notin \mathbf{R}$ is called a **head relation**

– α stands for an arbitrary SQL aggregation function

min, **max**, **count**, **countd**, **sum**, **avg**

– ϕ is a list of relational atoms over **R** called **condition**

– $[\psi]$ is a **nested condition**

– $\vec{x}$ is a sequence of **grouping variables** and $Var(q) \subseteq Var(\phi, \psi)$

DEF. By deleting aggregation functions and disregarding nestings we can transform an CAQ into its **core**

$$\tilde{q}(\vec{x}, \vec{y}) \leftarrow \phi, \psi.$$

Conditional Aggregate Queries (CAQs)

Let q be a CAQ of the form $q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$

Let $\vec{w} := Var(\phi, \psi)$ and $\vec{w}^{\phi} := \vec{w} \cap Var(\phi)$

Conditional Aggregate Queries (CAQs)

Let q be a CAQ of the form $q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$

Let $\vec{w} := Var(\phi, \psi)$ and $\vec{w}^{\phi} := \vec{w} \cap Var(\phi)$

DEF. An **assignment** γ over a condition ϕ maps variables in $Var(\phi, \psi)$ to tuples in a DB $\mathcal{D}$

Conditional Aggregate Queries (CAQs)

Let q be a CAQ of the form $q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$

Let $\vec{w} := Var(\phi, \psi)$ and $\vec{w}^{\phi} := \vec{w} \cap Var(\phi)$

DEF. An **assignment** γ over a condition ϕ maps variables in $Var(\phi, \psi)$ to tuples in a DB $\mathcal{D}$

DEF. We set of **satisfying assignments** and of **restricted satisfying** assignments to $\vec{w}^{\phi}$ and get

$$\begin{aligned} Sat_{\mathcal{D}}(\phi, \psi) &:= \{\gamma \mid \gamma \text{ satisfies } \phi \text{ and } \psi \text{ w.r.t. } \mathcal{D}\} \\ Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi, \psi) &:= \{\gamma \restriction \vec{w}^{\phi} \mid \gamma \in Sat_{\mathcal{D}}(\phi, \psi)\} \end{aligned}$$

Conditional Aggregate Queries (CAQs)

Let q be a CAQ of the form $q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$

Let $\vec{w} := Var(\phi, \psi)$ and $\vec{w}^{\phi} := \vec{w} \cap Var(\phi)$

DEF. An **assignment** γ over a condition ϕ maps variables in $Var(\phi, \psi)$ to tuples in a DB $\mathcal{D}$

DEF. We set of **satisfying assignments** and of **restricted satisfying** assignments to $\vec{w}^{\phi}$ and get

$$\begin{aligned} Sat_{\mathcal{D}}(\phi, \psi) &:= \{\gamma \mid \gamma \text{ satisfies } \phi \text{ and } \psi \text{ w.r.t. } \mathcal{D}\} \\ Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi, \psi) &:= \{\gamma \restriction \vec{w}^{\phi} \mid \gamma \in Sat_{\mathcal{D}}(\phi, \psi)\} \end{aligned}$$

SQL aggregation functions are given the standard SQL semantics: they are defined over **multisets** (bags) of values and return a number

Conditional Aggregate Queries (CAQs)

Let q be a CAQ of the form $q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$

Let $\vec{w} := Var(\phi, \psi)$ and $\vec{w}^{\phi} := \vec{w} \cap Var(\phi)$

DEF. An **assignment** γ over a condition ϕ maps variables in $Var(\phi, \psi)$ to tuples in a DB $\mathcal{D}$

DEF. We set of **satisfying assignments** and of **restricted satisfying** assignments to $\vec{w}^{\phi}$ and get

$$\begin{aligned} Sat_{\mathcal{D}}(\phi, \psi) &:= \{\gamma \mid \gamma \text{ satisfies } \phi \text{ and } \psi \text{ w.r.t. } \mathcal{D}\} \\ Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi, \psi) &:= \{\gamma \restriction \vec{w}^{\phi} \mid \gamma \in Sat_{\mathcal{D}}(\phi, \psi)\} \end{aligned}$$

SQL aggregation functions are given the standard SQL semantics: they are defined over **multisets** (bags) of values and return a number

DEF. A **group** of a tuple $\vec{d}$ is the multiset

$$G_{\vec{d}} := \{\!|\gamma(y) \mid \vec{d} = \gamma(\vec{x}) \text{ and } \gamma \in Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi, \psi)|\!\}$$

Conditional Aggregate Queries (CAQs)

Let q be a CAQ of the form $q(\vec{x}, \alpha(\vec{y})) \leftarrow \phi, [\psi]$

Let $\vec{w} := Var(\phi, \psi)$ and $\vec{w}^{\phi} := \vec{w} \cap Var(\phi)$

DEF. An **assignment** γ over a condition ϕ maps variables in $Var(\phi, \psi)$ to tuples in a DB $\mathcal{D}$

DEF. We set of **satisfying assignments** and of **restricted satisfying** assignments to $\vec{w}^{\phi}$ and get

$$\begin{aligned} Sat_{\mathcal{D}}(\phi, \psi) &:= \{\gamma \mid \gamma \text{ satisfies } \phi \text{ and } \psi \text{ w.r.t. } \mathcal{D}\} \\ Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi, \psi) &:= \{\gamma \restriction \vec{w}^{\phi} \mid \gamma \in Sat_{\mathcal{D}}(\phi, \psi)\} \end{aligned}$$

SQL aggregation functions are given the standard SQL semantics: they are defined over **multisets** (bags) of values and return a number

DEF. A **group** of a tuple $\vec{d}$ is the multiset

$$G_{\vec{d}} := \{\!|\gamma(y) \mid \vec{d} = \gamma(\vec{x}) \text{ and } \gamma \in Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi, \psi)|\!\}$$

DEF. The set of **answers** of q is defined as

$$q^{\mathcal{D}} := \{\langle \vec{d}, \alpha(G_{\vec{d}}) \rangle \mid \vec{d} = \gamma(\vec{x}) \text{ for some } \gamma \in Sat_{\mathcal{D}}^{\vec{w}^{\phi}}(\phi)\}$$

Conditional Aggregate Queries (CAQs)

EXAMPLE. Consider question **(b)** "how many parents own a car"

Conditional Aggregate Queries (CAQs)

EXAMPLE. Consider question **(b)** "how many parents own a car"

We can express **(b)** with the CAQ q_b

$$q_b(\textbf{count}) \leftarrow \mathit{Parent}(x), [\mathit{hasCar}(x, y)]$$

of core $\tilde{q}_b$

$$q_b(x) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

over the schema $\mathbf{R}_{\text{fam}} := \{\mathit{Parent}^1, \mathit{Child}^2, \mathit{Car}^1, \mathit{hasCar}^2, \mathit{hasChild}^2, \mathit{CTR}^2\}$

Conditional Aggregate Queries (CAQs)

EXAMPLE. Consider question **(b)** "how many parents own a car"

We can express **(b)** with the CAQ q_b

$$q_b(\textbf{count}) \leftarrow \mathit{Parent}(x), [\mathit{hasCar}(x, y)]$$

of core $\tilde{q}_b$

$$q_b(x) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

over the schema $\mathbf{R}_{\text{fam}} := \{\mathit{Parent}^1, \mathit{Child}^2, \mathit{Car}^1, \mathit{hasCar}^2, \mathit{hasChild}^2, \mathit{CTR}^2\}$

In SQL we would write a **nested** query

```
SELECT COUNT(*)
FROM Parent
WHERE EXISTS (SELECT *
        FROM Parent, hasCar)
```

Conditional Aggregate Queries (CAQs)

EXAMPLE. Consider question **(b)** "how many parents own a car"

We can express **(b)** with the CAQ q_b

$$q_b(\textbf{count}) \leftarrow \mathit{Parent}(x), [\mathit{hasCar}(x, y)]$$

of core $\tilde{q}_b$

$$q_b(x) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

over the schema $\mathbf{R}_{\text{fam}} := \{\mathit{Parent}^1, \mathit{Child}^2, \mathit{Car}^1, \mathit{hasCar}^2, \mathit{hasChild}^2, \mathit{CTR}^2\}$

In SQL we would write a **nested** query

```
SELECT COUNT(*)
FROM Parent
WHERE EXISTS (SELECT *
        FROM Parent, hasCar)
```

Consider the DB $\mathcal{D}_{\text{fam}}$ of $\mathbf{R}_{\text{fam}}$:

hasCar

PName	CType
Anna	VwGolf
Anna	FiatUno

Parent

PName
Anna

Conditional Aggregate Queries (CAQs)

EXAMPLE. Consider question **(b)** "how many parents own a car"

We can express **(b)** with the CAQ q_b

$$q_b(\textbf{count}) \leftarrow \mathit{Parent}(x), [\mathit{hasCar}(x, y)]$$

of core $\tilde{q}_b$

$$q_b(x) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

over the schema $\mathbf{R}_{\text{fam}} := \{\mathit{Parent}^1, \mathit{Child}^2, \mathit{Car}^1, \mathit{hasCar}^2, \mathit{hasChild}^2, \mathit{CTR}^2\}$

In SQL we would write a **nested** query

```
SELECT COUNT(*)
FROM Parent
WHERE EXISTS (SELECT *
        FROM Parent, hasCar)
```

Consider the DB $\mathcal{D}_{\text{fam}}$ of $\mathbf{R}_{\text{fam}}$:

hasCar

PName	CType
Anna	VwGolf
Anna	FiatUno

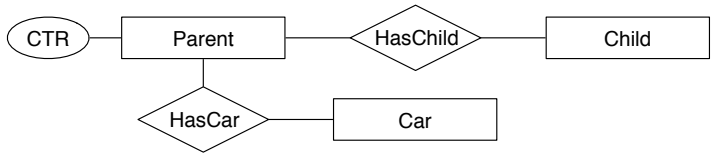
Parent

PName
Anna

Then: *(i)* $G_{\vec{\epsilon}} = \{\!|\langle \texttt{Anna} \rangle|\!\}$, *(ii)* $q_b^{\mathcal{D}_{\text{fam}}} = \{\langle 1 \rangle\}$ and *(iii)* $\tilde{q}_b^{\mathcal{D}_{\text{fam}}} = \{\langle \texttt{Anna} \rangle\}$

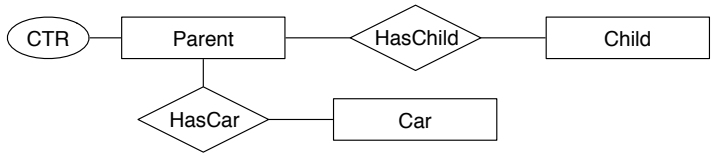
EXAMPLE. The ontology $\mathcal{T}_{\mathsf{fam}}$ encodes in *DL-Lite* the **Family** ontology:

EXAMPLE. The ontology $\mathcal{T}_{\text{fam}}$ encodes in *DL-Lite* the **Family** ontology:



$$\begin{array}{ll} Parent \sqsubseteq \exists CTR & \exists CTR \sqsubseteq Parent \\ & \exists hasChild \sqsubseteq Parent \\ Parent \sqsubseteq \exists hasCar & \exists hasCar \sqsubseteq Parent \\ \exists hasCar^- \sqsubseteq Car & \exists hasChild^- \sqsubseteq Child \end{array}$$

EXAMPLE. The ontology $\mathcal{T}_{\text{fam}}$ encodes in *DL-Lite* the **Family** ontology:



$$\begin{array}{ll} \mathit{Parent} \sqsubseteq \exists \mathit{CTR} & \exists \mathit{CTR} \sqsubseteq \mathit{Parent} \\ & \exists \mathit{hasChild} \sqsubseteq \mathit{Parent} \\ \mathit{Parent} \sqsubseteq \exists \mathit{hasCar} & \exists \mathit{hasCar} \sqsubseteq \mathit{Parent} \\ \exists \mathit{hasCar}^- \sqsubseteq \mathit{Car} & \exists \mathit{hasChild}^- \sqsubseteq \mathit{Child} \end{array}$$

Whereas $\mathcal{D}_{\text{fam}}$ now stores data consistent with the $\mathcal{T}_{\text{fam}}$ ontology:

Parent

PName
Anna

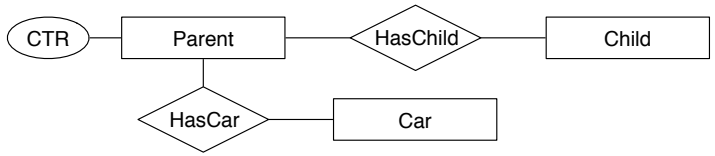
CTR

PName	Amount
Luisa	100
Anna	150

hasCar

PName	CType
Anna	VwGolf
Anna	FiatUno

EXAMPLE. The ontology $\mathcal{T}_{\text{fam}}$ encodes in *DL-Lite* the **Family** ontology:



$$\begin{array}{ll} \mathit{Parent} \sqsubseteq \exists \mathit{CTR} & \exists \mathit{CTR} \sqsubseteq \mathit{Parent} \\ & \exists \mathit{hasChild} \sqsubseteq \mathit{Parent} \\ \mathit{Parent} \sqsubseteq \exists \mathit{hasCar} & \exists \mathit{hasCar} \sqsubseteq \mathit{Parent} \\ \exists \mathit{hasCar}^- \sqsubseteq \mathit{Car} & \exists \mathit{hasChild}^- \sqsubseteq \mathit{Child} \end{array}$$

Whereas $\mathcal{D}_{\text{fam}}$ now stores data consistent with the $\mathcal{T}_{\text{fam}}$ ontology:

Parent

PName
Anna

CTR

PName	Amount
Luisa	100
Anna	150

hasCar

PName	CType
Anna	VwGolf
Anna	FiatUno

By combining them we get an OBDAS $\langle \mathcal{T}_{\text{fam}}, \mathcal{D}_{\text{fam}} \rangle$ over the family domain

Certain Answers and CQs

DEF. The **certain answers** of a CQ q over an OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$ are defined as

$$cert(q, \mathcal{D}, \mathcal{T}) := \bigcap \{ q^{\mathcal{D}'} \mid \mathcal{D}' \models \langle \mathcal{D}, \mathcal{T} \rangle \}.$$

Certain Answers and CQs

DEF. The **certain answers** of a CQ q over an OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$ are defined as

$$cert(q, \mathcal{D}, \mathcal{T}) := \bigcap \{ q^{\mathcal{D}'} \mid \mathcal{D}' \models \langle \mathcal{D}, \mathcal{T} \rangle \}.$$

We pick tuples that are answers to q over **all** the possible DBs $\mathcal{D}'$ compatible with a OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$

Certain Answers and CQs

DEF. The **certain answers** of a CQ q over an OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$ are defined as

$$cert(q, \mathcal{D}, \mathcal{T}) := \bigcap \{ q^{\mathcal{D}'} \mid \mathcal{D}' \models \langle \mathcal{D}, \mathcal{T} \rangle \}.$$

We pick tuples that are answers to q over **all** the possible DBs $\mathcal{D}'$ compatible with a OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$

The certain answers of CQs over DL-Lite ontologies are **always defined** and can be computed efficiently [Calvanese et al. 2007]

Certain Answers and CQs

DEF. The **certain answers** of a CQ q over an OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$ are defined as

$$cert(q, \mathcal{D}, \mathcal{T}) := \bigcap \{ q^{\mathcal{D}'} \mid \mathcal{D}' \models \langle \mathcal{D}, \mathcal{T} \rangle \}.$$

We pick tuples that are answers to q over **all** the possible DBs $\mathcal{D}'$ compatible with a OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$

The certain answers of CQs over DL-Lite ontologies are **always defined** and can be computed efficiently [Calvanese et al. 2007]

EXAMPLE. Consider as CQ (the core of q_b) that asks for all parents that own cars:

$$\tilde{q}_b(x) \leftarrow Parent(x), hasCar(x, y).$$

Asking $\tilde{q}_b$ to $\langle \mathcal{T}_{\text{fam}}, \mathcal{D}_{\text{fam}} \rangle$ gives as certain answers

$$cert(\tilde{q}_b, \mathcal{T}_{\text{fam}}, \mathcal{D}_{\text{fam}}) = \{ \langle \texttt{Anna} \rangle, \langle \texttt{Luisa} \rangle \}$$

Certain Answers and CQs

DEF. The **certain answers** of a CQ q over an OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$ are defined as

$$cert(q, \mathcal{D}, \mathcal{T}) := \bigcap \{ q^{\mathcal{D}'} \mid \mathcal{D}' \models \langle \mathcal{D}, \mathcal{T} \rangle \}.$$

We pick tuples that are answers to q over **all** the possible DBs $\mathcal{D}'$ compatible with a OBDAS $\langle \mathcal{D}, \mathcal{T} \rangle$

The certain answers of CQs over DL-Lite ontologies are **always defined** and can be computed efficiently [Calvanese et al. 2007]

EXAMPLE. Consider as CQ (the core of q_b) that asks for all parents that own cars:

$$\tilde{q}_b(x) \leftarrow Parent(x), hasCar(x, y).$$

Asking $\tilde{q}_b$ to $\langle \mathcal{T}_{\text{fam}}, \mathcal{D}_{\text{fam}} \rangle$ gives as certain answers

$$cert(\tilde{q}_b, \mathcal{T}_{\text{fam}}, \mathcal{D}_{\text{fam}}) = \{ \langle \texttt{Anna} \rangle, \langle \texttt{Luisa} \rangle \}$$

This does not carry on to AQs

Certain Answers and AQs

THEO. The certain answers of CAQs over OBDASs are, in general, empty

Certain Answers and AQs

THEO. The certain answers of CAQs over OBDASs are, in general, empty

PROOF. Consider the CAQ q'_b for **(b)**

$$q'_b(\mathbf{countd}(x)) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

Certain Answers and AQs

THEO. The certain answers of CAQs over OBDASs are, in general, empty

PROOF. Consider the CAQ q'_b for **(b)**

$$q'_b(\mathbf{countd}(x)) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

In different DBs, multiplicities can differ: consider the following two DBs $\mathcal{D}'$ and $\mathcal{D}''$ satisfying $\langle \mathcal{T}_{\mathsf{fam}}, \mathcal{D}_{\mathsf{fam}} \rangle$ where

Certain Answers and AQs

THEO. The certain answers of CAQs over OBDASs are, in general, empty

PROOF. Consider the CAQ q'_b for **(b)**

$$q'_b(\mathbf{countd}(x)) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

In different DBs, multiplicities can differ: consider the following two DBs $\mathcal{D}'$ and $\mathcal{D}''$ satisfying $\langle \mathcal{T}_{\mathsf{fam}}, \mathcal{D}_{\mathsf{fam}} \rangle$ where

(i) $\mathcal{D}' := \mathcal{D}_{\mathsf{fam}}$ and

Certain Answers and AQs

THEO. The certain answers of CAQs over OBDASs are, in general, empty

PROOF. Consider the CAQ q'_b for **(b)**

$$q'_b(\mathbf{countd}(x)) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

In different DBs, multiplicities can differ: consider the following two DBs $\mathcal{D}'$ and $\mathcal{D}''$ satisfying $\langle \mathcal{T}_{\mathsf{fam}}, \mathcal{D}_{\mathsf{fam}} \rangle$ where

(i) $\mathcal{D}' := \mathcal{D}_{\mathsf{fam}}$ and

(ii) $\mathcal{D}''$ is obtained from $\mathcal{D}_{\mathsf{fam}}$ by putting

$$\mathit{hasCar}^{\mathcal{D}''} := \mathit{hasCar}^{\mathcal{D}_{\mathsf{fam}}} \cup \{\langle \mathtt{Luisa}, \mathtt{Smart} \rangle\}$$

THEO. The certain answers of CAQs over OBDASs are, in general, empty

PROOF. Consider the CAQ q'_b for **(b)**

$$q'_b(\mathbf{countd}(x)) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

In different DBs, multiplicities can differ: consider the following two DBs $\mathcal{D}'$ and $\mathcal{D}''$ satisfying $\langle \mathcal{T}_{\mathsf{fam}}, \mathcal{D}_{\mathsf{fam}} \rangle$ where

(i) $\mathcal{D}' := \mathcal{D}_{\mathsf{fam}}$ and

(ii) $\mathcal{D}''$ is obtained from $\mathcal{D}_{\mathsf{fam}}$ by putting

$$\mathit{hasCar}^{\mathcal{D}''} := \mathit{hasCar}^{\mathcal{D}_{\mathsf{fam}}} \cup \{\langle \mathtt{Luisa}, \mathtt{Smart} \rangle\}$$

Then, ${q'_b}^{\mathcal{D}''} = \{\langle 2 \rangle\}$, $q_b^{\mathcal{D}'} = \{\langle 1 \rangle\}$ and

$${q'_b}^{\mathcal{D}''} \cap q_b^{\mathcal{D}'} = \emptyset.$$

THEO. The certain answers of CAQs over OBDASs are, in general, empty

PROOF. Consider the CAQ q'_b for **(b)**

$$q'_b(\mathbf{countd}(x)) \leftarrow \mathit{Parent}(x), \mathit{hasCar}(x, y)$$

In different DBs, multiplicities can differ: consider the following two DBs $\mathcal{D}'$ and $\mathcal{D}''$ satisfying $\langle \mathcal{T}_{\mathsf{fam}}, \mathcal{D}_{\mathsf{fam}} \rangle$ where

(i) $\mathcal{D}' := \mathcal{D}_{\mathsf{fam}}$ and

(ii) $\mathcal{D}''$ is obtained from $\mathcal{D}_{\mathsf{fam}}$ by putting

$$\mathit{hasCar}^{\mathcal{D}''} := \mathit{hasCar}^{\mathcal{D}_{\mathsf{fam}}} \cup \{\langle \mathtt{Luisa}, \mathtt{Smart} \rangle\}$$

Then, $q'^{\mathcal{D}''}_b = \{\langle 2 \rangle\}$, $q^{\mathcal{D}'}_b = \{\langle 1 \rangle\}$ and

$$q'^{\mathcal{D}''}_b \cap q^{\mathcal{D}'}_b = \emptyset.$$

Therefore:

$$\mathit{cert}(q_b, \mathcal{T}_{\mathsf{fam}}, \mathcal{D}_{\mathsf{fam}}) = \emptyset$$

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

– **K**d variables give rise to the same multiplicities over **all** DBs

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

– **K**d variables give rise to the same multiplicities over **all** DBs

– $\vec{x} \cup \vec{y} \cup \vec{z} \subseteq Var(\phi, \psi)$

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

– **K**d variables give rise to the same multiplicities over **all** DBs

– $\vec{x} \cup \vec{y} \cup \vec{z} \subseteq Var(\phi, \psi)$

– bracketing ($= [\cdot]$) collapses multiplicities in conditions

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

– **K**d variables give rise to the same multiplicities over **all** DBs

– $\vec{x} \cup \vec{y} \cup \vec{z} \subseteq Var(\phi, \psi)$

– bracketing ($= [\cdot]$) collapses multiplicities in conditions

– non-**K**d variables are DB-dependant

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

– **K**d variables give rise to the same multiplicities over **all** DBs

– $\vec{x} \cup \vec{y} \cup \vec{z} \subseteq Var(\phi, \psi)$

– bracketing ($= [\cdot]$) collapses multiplicities in conditions

– non-**K**d variables are DB-dependant

DEF. The **auxiliary query** of an EAQ is

$$q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$$

Epistemic Aggregate Queries (EAQs)

We can overcome this problem by asking about **known** tuples:

DEF. An **epistemic aggregate query** is a query of the form

$$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi].$$

– **K** is an **epistemic** operator (**known** variables)

– **K**d variables give rise to the same multiplicities over **all** DBs

– $\vec{x} \cup \vec{y} \cup \vec{z} \subseteq Var(\phi, \psi)$

– bracketing ($= [\cdot]$) collapses multiplicities in conditions

– non-**K**d variables are DB-dependant

DEF. The **auxiliary query** of an EAQ is

$$q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$$

EAQs extend the syntax and semantics of AQs

Epistemic Aggregate Queries (EAQs)

Consider an ODBAS $\mathcal{K} := \langle \mathcal{O}, \mathcal{D} \rangle$, $\mathcal{D}'$ a DB s.t. $\mathcal{D}' \models \mathcal{K}$ and an EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\phi, [\psi]$ with auxiliary query $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$

Let $\vec{w} := \vec{x} \cup \vec{y} \cup \vec{z}$ and $\vec{z}^{\phi} := \vec{z} \cap \mathit{Var}(\phi)$

Epistemic Aggregate Queries (EAQs)

Consider an ODBAS $\mathcal{K} := \langle \mathcal{O}, \mathcal{D} \rangle$, $\mathcal{D}'$ a DB s.t. $\mathcal{D}' \models \mathcal{K}$ and an EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\phi, [\psi]$ with auxiliary query $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$

Let $\vec{w} := \vec{x} \cup \vec{y} \cup \vec{z}$ and $\vec{z}^{\phi} := \vec{z} \cap Var(\phi)$

DEF. The **K satisfying assignments** and the **K restricted satisfying assignments** are defined as

$$\begin{aligned} KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi) &:= \{\gamma \in Sat_{\mathcal{D}'}(\phi, \psi) \mid \gamma(\vec{w}) \in cert(q_{aux}, \mathcal{K})\} \\ KSat^{\vec{x} \cup \vec{y} \cup \vec{z}^{\phi}}_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi) &:= \{\gamma \upharpoonright \vec{x} \cup \vec{y} \cup \vec{z}^{\phi} \mid \gamma \in KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi)\} \end{aligned}$$

Epistemic Aggregate Queries (EAQs)

Consider an ODBAS $\mathcal{K} := \langle \mathcal{O}, \mathcal{D} \rangle$, $\mathcal{D}'$ a DB s.t. $\mathcal{D}' \models \mathcal{K}$ and an EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\phi, [\psi]$ with auxiliary query $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$

Let $\vec{w} := \vec{x} \cup \vec{y} \cup \vec{z}$ and $\vec{z}^{\phi} := \vec{z} \cap Var(\phi)$

DEF. The **K satisfying assignments** and the **K restricted satisfying assignments** are defined as

$$\begin{array}{rcl} KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi) & := & \{\gamma \in Sat_{\mathcal{D}'}(\phi, \psi) \mid \gamma(\vec{w}) \in cert(q_{aux}, \mathcal{K})\} \\ KSat^{\vec{x}\cup\vec{y}\cup\vec{z}^{\phi}}_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi) & := & \{\gamma \upharpoonright \vec{x} \cup \vec{y} \cup \vec{z}^{\phi} \mid \gamma \in KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi)\} \end{array}$$

DEF. The **K group** of tuple $\vec{d}$ is defined as

$$H_{\vec{d}} := \{\!|\gamma(\vec{y}) \mid \gamma(\vec{x}) = \vec{d} \text{ and } \gamma \in KSat^{\vec{x}\cup\vec{y}\cup\vec{z}^{\phi}}_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi)|\!\}$$

Epistemic Aggregate Queries (EAQs)

Consider an ODBAS $\mathcal{K} := \langle \mathcal{O}, \mathcal{D} \rangle$, $\mathcal{D}'$ a DB s.t. $\mathcal{D}' \models \mathcal{K}$ and an EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\phi, [\psi]$ with auxiliary query $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$

Let $\vec{w} := \vec{x} \cup \vec{y} \cup \vec{z}$ and $\vec{z}^{\phi} := \vec{z} \cap Var(\phi)$

DEF. The **K satisfying assignments** and the **K restricted satisfying assignments** are defined as

$$\begin{array}{rcl} KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi) & := & \{\gamma \in Sat_{\mathcal{D}'}(\phi, \psi) \mid \gamma(\vec{w}) \in cert(q_{aux}, \mathcal{K})\} \\ KSat^{\vec{x} \cup \vec{y} \cup \vec{z}^{\phi}}_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi) & := & \{\gamma \upharpoonright \vec{x} \cup \vec{y} \cup \vec{z}^{\phi} \mid \gamma \in KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi)\} \end{array}$$

DEF. The **K group** of tuple $\vec{d}$ is defined as

$$H_{\vec{d}} := \{\!|\gamma(\vec{y}) \mid \gamma(\vec{x}) = \vec{d} \text{ and } \gamma \in KSat^{\vec{x} \cup \vec{y} \cup \vec{z}^{\phi}}_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi)|\!\}$$

DEF. The **K answers** of q over $\mathcal{D}'$ and the **epistemic certain answers** of q over $\mathcal{K}$ are defined as

$$\begin{array}{rcl} q(\mathcal{D}', \mathcal{K}) & := & \{\langle \vec{d}, \alpha(H_{\vec{d}}) \rangle \mid \gamma(\vec{x}) = \vec{d} \text{ for some } \gamma \in KSat_{\mathcal{D}',\mathcal{K}}(\vec{w}; \phi, \psi)\} \\ Ecert(q, \mathcal{K}) & := & \bigcap \{q(\mathcal{D}', \mathcal{K}) \mid \mathcal{D}' \models \mathcal{K}\} \end{array}$$

Epistemic Aggregate Queries (EAQs)

EXAMPLE. Consider the EAQ q_b^k for **(b)**

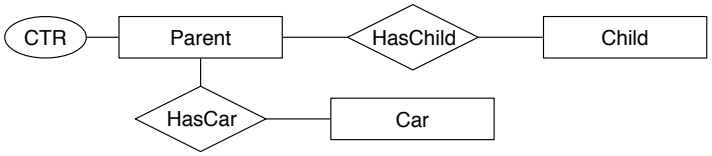
$$q_b^k(\mathbf{countd}(x)) \leftarrow \mathbf{K}x, y.Parent(x), hasCar(x, y)$$

Epistemic Aggregate Queries (EAQs)

EXAMPLE. Consider the EAQ q_b^k for **(b)**

$$q_b^k(\textbf{countd}(x)) \leftarrow \mathbf{K}x, y.Parent(x), hasCar(x, y)$$

That asks how many parents are known to own a car by the OBDAS $\langle \mathcal{T}_{fam}, \mathcal{D}_{fam} \rangle$:



Parent

PName
Anna

CTR

PName	Amount
Luisa	100
Anna	150

hasCar

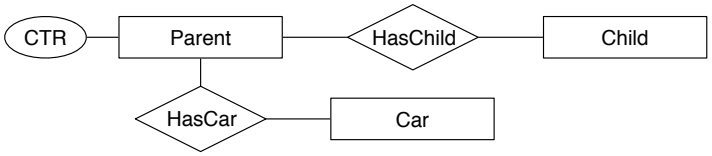
PName	CType
Anna	VwGolf
Anna	FiatUno

Epistemic Aggregate Queries (EAQs)

EXAMPLE. Consider the EAQ q_b^k for **(b)**

$$q_b^k(\textbf{countd}(x)) \leftarrow \mathbf{K}x, y.Parent(x), hasCar(x, y)$$

That asks how many parents are known to own a car by the OBDAS $\langle \mathcal{T}_{fam}, \mathcal{D}_{fam} \rangle$:



Parent
PName
Anna

CTR	
PName	Amount
Luisa	100
Anna	150

hasCar	
PName	CType
Anna	VwGolf
Anna	FiatUno

Since we now restrict ourselves to information explicitly stated in the ODBAS, we get

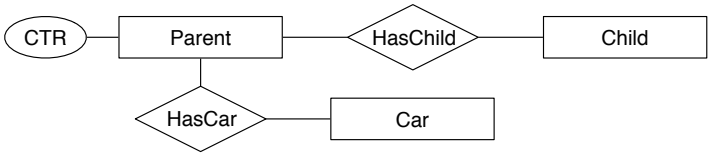
$$H_{\vec{\epsilon}} = \{\!|\langle \texttt{Anna} \rangle, \langle \texttt{Anna} \rangle|\!\}$$

Epistemic Aggregate Queries (EAQs)

EXAMPLE. Consider the EAQ q_b^k for **(b)**

$$q_b^k(\textbf{countd}(x)) \leftarrow \mathbf{K}x, y.Parent(x), hasCar(x, y)$$

That asks how many parents are known to own a car by the OBDAS $\langle \mathcal{T}_{fam}, \mathcal{D}_{fam} \rangle$:



Parent
PName
Anna

CTR	
PName	Amount
Luisa	100
Anna	150

hasCar	
PName	CType
Anna	VwGolf
Anna	FiatUno

Since we now restrict ourselves to information explicitly stated in the ODBAS, we get

$$H_{\vec{\epsilon}} = \{\!|\langle \texttt{Anna} \rangle, \langle \texttt{Anna} \rangle|\!\}$$

Therefore, $\textbf{countd}(\{\!|\texttt{Anna}, \texttt{Anna}|\!\}) = 1$ and the epistemic certain answers are

$$ECert(q_b^k, \mathcal{T}_{fam}, \mathcal{D}_{fam}) = \{\langle 1 \rangle\}$$

The Generic Algorithm GA

Input:	EAQ	$q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$
	DL-Lite ODBAS	$\mathcal{K}$
Output:	set of tuples	O

The Generic Algorithm GA

Input: EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$

DL-Lite ODBAS $\mathcal{K}$

Output: set of tuples O

Do: build q_{aux}: $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$

build q_0: $v_0(\vec{x}, \vec{y}, \vec{z}^{\phi}) \leftarrow Cert(aux_q, \mathcal{K})(\vec{x}, \vec{y}, \vec{z})$

build q_1: $q_1(\vec{x}, \alpha(y)) \leftarrow v_0(\vec{x}, \vec{y}, \vec{z}^{\phi})$

compute certain answers: $D_1 := cert(aux_q, \langle \mathcal{D}, \mathcal{T} \rangle)$

project on **K**-variables: $D_2 := v_0^{D_1}$

aggregate: $O := q_1^{D_2}$

The Generic Algorithm GA

Input: EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$
DL-Lite ODBAS $\mathcal{K}$
Output: set of tuples O

Do: build q_{aux}: $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$
build q_0: $v_0(\vec{x}, \vec{y}, \vec{z}^{\phi}) \leftarrow Cert(aux_q, \mathcal{K})(\vec{x}, \vec{y}, \vec{z})$
build q_1: $q_1(\vec{x}, \alpha(y)) \leftarrow v_0(\vec{x}, \vec{y}, \vec{z}^{\phi})$
compute certain answers: $D_1 := cert(aux_q, \langle \mathcal{D}, \mathcal{T} \rangle)$
project on **K**-variables: $D_2 := v_0^{D_1}$
aggregate: $O := q_1^{D_2}$

– v_0 is a **view** that stores the certain answers of q_{aux}

The Generic Algorithm GA

Input: EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$
DL-Lite ODBAS $\mathcal{K}$
Output: set of tuples O

Do: build q_{aux}: $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$
build q_0: $v_0(\vec{x}, \vec{y}, \vec{z}^{\phi}) \leftarrow Cert(aux_q, \mathcal{K})(\vec{x}, \vec{y}, \vec{z})$
build q_1: $q_1(\vec{x}, \alpha(y)) \leftarrow v_0(\vec{x}, \vec{y}, \vec{z}^{\phi})$
compute certain answers: $D_1 := cert(aux_q, \langle \mathcal{D}, \mathcal{T} \rangle)$
project on **K**-variables: $D_2 := v_0^{D_1}$
aggregate: $O := q_1^{D_2}$

– v_0 is a **view** that stores the certain answers of q_{aux}

– $Cert(aux_q, \mathcal{K})$ is an ad hoc **predicate** that depends on $\mathcal{K}$ and q

The Generic Algorithm GA

Input: EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$
DL-Lite ODBAS $\mathcal{K}$
Output: set of tuples O

Do: build q_{aux}: $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$
build q_0: $v_0(\vec{x}, \vec{y}, \vec{z}^{\phi}) \leftarrow Cert(aux_q, \mathcal{K})(\vec{x}, \vec{y}, \vec{z})$
build q_1: $q_1(\vec{x}, \alpha(y)) \leftarrow v_0(\vec{x}, \vec{y}, \vec{z}^{\phi})$
compute certain answers: $D_1 := cert(aux_q, \langle \mathcal{D}, \mathcal{T} \rangle)$
project on **K**-variables: $D_2 := v_0^{D_1}$
aggregate: $O := q_1^{D_2}$

– v_0 is a **view** that stores the certain answers of q_{aux}

– $Cert(aux_q, \mathcal{K})$ is an ad hoc **predicate** that depends on $\mathcal{K}$ and q

– $\vec{z}^{\phi} := \vec{z} \cap Var(\phi)$ **projects** away bracketed variables (we disregard multiplicities)

The Generic Algorithm GA

Input: EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$
DL-Lite ODBAS $\mathcal{K}$
Output: set of tuples O

Do: build q_{aux}: $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$
build q_0: $v_0(\vec{x}, \vec{y}, \vec{z}^{\phi}) \leftarrow Cert(aux_q, \mathcal{K})(\vec{x}, \vec{y}, \vec{z})$
build q_1: $q_1(\vec{x}, \alpha(y)) \leftarrow v_0(\vec{x}, \vec{y}, \vec{z}^{\phi})$
compute certain answers: $D_1 := cert(aux_q, \langle \mathcal{D}, \mathcal{T} \rangle)$
project on **K**-variables: $D_2 := v_0^{D_1}$
aggregate: $O := q_1^{D_2}$

– v_0 is a **view** that stores the certain answers of q_{aux}
– $Cert(aux_q, \mathcal{K})$ is an ad hoc **predicate** that depends on $\mathcal{K}$ and q
– $\vec{z}^{\phi} := \vec{z} \cap Var(\phi)$ **projects** away bracketed variables (we disregard multiplicities)
– q_1 **aggregates** on top of the certain answers

The Generic Algorithm GA

Input: EAQ $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$
DL-Lite ODBAS $\mathcal{K}$
Output: set of tuples O

Do: build q_{aux}: $q_{aux}(\vec{x}, \vec{y}, \vec{z}) \leftarrow \phi, \psi$
build q_0: $v_0(\vec{x}, \vec{y}, \vec{z}^{\phi}) \leftarrow Cert(aux_q, \mathcal{K})(\vec{x}, \vec{y}, \vec{z})$
build q_1: $q_1(\vec{x}, \alpha(y)) \leftarrow v_0(\vec{x}, \vec{y}, \vec{z}^{\phi})$
compute certain answers: $D_1 := cert(aux_q, \langle \mathcal{D}, \mathcal{T} \rangle)$
project on **K**-variables: $D_2 := v_0^{D_1}$
aggregate: $O := q_1^{D_2}$

– v_0 is a **view** that stores the certain answers of q_{aux}
– $Cert(aux_q, \mathcal{K})$ is an ad hoc **predicate** that depends on $\mathcal{K}$ and q
– $\vec{z}^{\phi} := \vec{z} \cap Var(\phi)$ **projects** away bracketed variables (we disregard multiplicities)
– q_1 **aggregates** on top of the certain answers

GA is sound and complete w.r.t. $Ecert(q, \mathcal{K})$ under certain conditions

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in \mathit{Var}(\phi, \psi)$

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in Var(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $Ecert(q, \mathcal{O}, \mathcal{D}) = \emptyset$

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in Var(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $Ecert(q, \mathcal{O}, \mathcal{D}) = \emptyset$

DEF. EAQ q is said to be **coherent** with $\mathcal{K}$ if there exists $\mathcal{D}'$ s.t. $\mathcal{D}' \models \mathcal{D}$ and $q(\mathcal{D}, \mathcal{K}) \neq \emptyset$

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in Var(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $Ecert(q, \mathcal{O}, \mathcal{D}) = \emptyset$

DEF. EAQ q is said to be **coherent** with $\mathcal{K}$ if there exists $\mathcal{D}'$ s.t. $\mathcal{D}' \models \mathcal{D}$ and $q(\mathcal{D}, \mathcal{K}) \neq \emptyset$

DEF. Variable y is said to **depend** on variable x if either *(i)* $(\mathsf{funct}R) \in \mathcal{T}$ and $R(x, y)$ occurs in ϕ, ψ, or *(ii)* $(\mathsf{funct}R^-) \in \mathcal{T}$ and $R(y, x)$ occurs in ϕ, ψ

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in \mathit{Var}(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $\mathit{Ecert}(q, \mathcal{O}, \mathcal{D}) = \emptyset$

DEF. EAQ q is said to be **coherent** with $\mathcal{K}$ if there exists $\mathcal{D}'$ s.t. $\mathcal{D}' \models \mathcal{D}$ and $q(\mathcal{D}, \mathcal{K}) \neq \emptyset$

DEF. Variable y is said to **depend** on variable x if either *(i)* $(\mathsf{funct}R) \in \mathcal{T}$ and $R(x, y)$ occurs in ϕ, ψ, or *(ii)* $(\mathsf{funct}R^-) \in \mathcal{T}$ and $R(y, x)$ occurs in ϕ, ψ

DEF. The relation of **functional dependence** is the reflexive and transitive closure of variable dependence

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in \mathit{Var}(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $\mathit{Ecert}(q, \mathcal{O}, \mathcal{D}) = \emptyset$

DEF. EAQ q is said to be **coherent** with $\mathcal{K}$ if there exists $\mathcal{D}'$ s.t. $\mathcal{D}' \models \mathcal{D}$ and $q(\mathcal{D}, \mathcal{K}) \neq \emptyset$

DEF. Variable y is said to **depend** on variable x if either *(i)* $(\mathsf{funct}R) \in \mathcal{T}$ and $R(x, y)$ occurs in ϕ, ψ, or *(ii)* $(\mathsf{funct}R^-) \in \mathcal{T}$ and $R(y, x)$ occurs in ϕ, ψ

DEF. The relation of **functional dependence** is the reflexive and transitive closure of variable dependence

DEF. Variable x is said to be **restricted** if it is **K**d or functionally depends on a restricted variable

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in \mathit{Var}(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $\mathit{Ecert}(q, \mathcal{O}, \mathcal{D}) = \emptyset$

DEF. EAQ q is said to be **coherent** with $\mathcal{K}$ if there exists $\mathcal{D}'$ s.t. $\mathcal{D}' \models \mathcal{D}$ and $q(\mathcal{D}, \mathcal{K}) \neq \emptyset$

DEF. Variable y is said to **depend** on variable x if either *(i)* $(\mathsf{funct}R) \in \mathcal{T}$ and $R(x, y)$ occurs in ϕ, ψ, or *(ii)* $(\mathsf{funct}R^-) \in \mathcal{T}$ and $R(y, x)$ occurs in ϕ, ψ

DEF. The relation of **functional dependence** is the reflexive and transitive closure of variable dependence

DEF. Variable x is said to be **restricted** if it is $\mathbf{K}$d or functionally depends on a restricted variable

DEF. EAQ q is said to be **restricted** if all its variables are restricted

Properties

Let $\mathcal{K} := \langle \mathcal{T}, \mathcal{D} \rangle$ be an ODBA, let $q(\vec{x}, \alpha(\vec{y})) \leftarrow \mathbf{K}\vec{x}, \vec{y}, \vec{z}.\ \phi, [\psi]$ be an EAQ and $x, y \in Var(\phi, \psi)$

DEF. EAQ q is said to be **trivial** w.r.t. $\mathcal{O}$ if for all $\mathcal{D}$, $Ecert(q, \mathcal{O}, \mathcal{D}) = \emptyset$

DEF. EAQ q is said to be **coherent** with $\mathcal{K}$ if there exists $\mathcal{D}'$ s.t. $\mathcal{D}' \models \mathcal{D}$ and $q(\mathcal{D}, \mathcal{K}) \neq \emptyset$

DEF. Variable y is said to **depend** on variable x if either *(i)* $(\text{funct}R) \in \mathcal{T}$ and $R(x, y)$ occurs in ϕ, ψ, or *(ii)* $(\text{funct}R^-) \in \mathcal{T}$ and $R(y, x)$ occurs in ϕ, ψ

DEF. The relation of **functional dependence** is the reflexive and transitive closure of variable dependence

DEF. Variable x is said to be **restricted** if it is $\mathbf{K}$d or functionally depends on a restricted variable

DEF. EAQ q is said to be **restricted** if all its variables are restricted

Restrictedness ensures non-triviality, soundness and completeness

Properties

EXAMPLE. Consider EAQ q_c^k for question **(c)**

$$q(x, \mathbf{sum}(y)) \leftarrow \mathbf{K}x, yParent(x), CTR(x, y), hasChild(x, z)$$

EXAMPLE. Consider EAQ q_c^k for question **(c)**

$$q(x, \mathbf{sum}(y)) \leftarrow \mathbf{K}x, y\, Parent(x), CTR(x, y), hasChild(x, z)$$

Consider again our **Family** OBDAS

$Parent \sqsubseteq \exists CTR$ $\quad \exists CTR \sqsubseteq Parent$

$\exists hasChild \sqsubseteq Parent$

$Parent \sqsubseteq \exists hasCar$ $\quad \exists hasCar \sqsubseteq Parent$

$\exists hasCar^- \sqsubseteq Car$ $\quad \exists hasChild^- \sqsubseteq Child$

Parent
PName
Anna

CTR	
PName	Amount
Luisa	100
Anna	150

hasCar	
PName	CType
Anna	VwGolf
Anna	FiatUno

Properties

EXAMPLE. Consider EAQ q_c^k for question **(c)**

$$q(x, \mathbf{sum}(y)) \leftarrow \mathbf{K}x, yParent(x), CTR(x, y), hasChild(x, z)$$

Consider again our **Family** OBDAS

$Parent \sqsubseteq \exists CTR$ $\quad \exists CTR \sqsubseteq Parent$
$\exists hasChild \sqsubseteq Parent$
$Parent \sqsubseteq \exists hasCar$ $\quad \exists hasCar \sqsubseteq Parent$
$\exists hasCar^- \sqsubseteq Car$ $\quad \exists hasChild^- \sqsubseteq Child$

Parent
PName
Anna

CTR	
PName	Amount
Luisa	100
Anna	150

hasCar	
PName	CType
Anna	VwGolf
Anna	FiatUno

(i) If we add to $\mathcal{T}_{fam}$ the assertions (funct *hasChild*) and $Parent \sqsubseteq \exists hasChild$, restricting z, then

$$Ecert(q_c^k, \mathcal{T}_{fam}, \mathcal{D}_{fam}) = \{\langle \mathtt{Luisa}, 100\rangle\langle \mathtt{Anne}, 150\rangle\}$$

EXAMPLE. Consider EAQ q_c^k for question **(c)**

$$q(x, \mathbf{sum}(y)) \leftarrow \mathbf{K}x, y\, Parent(x), CTR(x, y), hasChild(x, z)$$

Consider again our **Family** OBDAS

$Parent \sqsubseteq \exists CTR$ $\quad \exists CTR \sqsubseteq Parent$
$\exists hasChild \sqsubseteq Parent$
$Parent \sqsubseteq \exists hasCar$ $\quad \exists hasCar \sqsubseteq Parent$
$\exists hasCar^- \sqsubseteq Car$ $\quad \exists hasChild^- \sqsubseteq Child$

Parent
PName
Anna

CTR	
PName	Amount
Luisa	100
Anna	150

hasCar	
PName	CType
Anna	VwGolf
Anna	FiatUno

(i) If we add to $\mathcal{T}_{fam}$ the assertions (funct *hasChild*) and $Parent \sqsubseteq \exists hasChild$, restricting z, then

$$Ecert(q_c^k, \mathcal{T}_{fam}, \mathcal{D}_{fam}) = \{\langle \mathtt{Luisa}, 100\rangle \langle \mathtt{Anne}, 150\rangle\}$$

(ii) Otherwise

$$Ecert(q_c^k, \mathcal{T}_{fam}, \mathcal{D}_{fam}) = \emptyset$$

EXAMPLE. Consider EAQ q_c^k for question **(c)**

$$q(x, \mathbf{sum}(y)) \leftarrow \mathbf{K}x, y\, Parent(x), CTR(x, y), hasChild(x, z)$$

Consider again our **Family** OBDAS

$Parent \sqsubseteq \exists CTR$ $\quad \exists CTR \sqsubseteq Parent$
$\exists hasChild \sqsubseteq Parent$
$Parent \sqsubseteq \exists hasCar$ $\quad \exists hasCar \sqsubseteq Parent$
$\exists hasCar^- \sqsubseteq Car$ $\quad \exists hasChild^- \sqsubseteq Child$

Parent
PName
Anna

CTR	
PName	Amount
Luisa	100
Anna	150

hasCar	
PName	CType
Anna	VwGolf
Anna	FiatUno

(i) If we add to $\mathcal{T}_{fam}$ the assertions (funct *hasChild*) and $Parent \sqsubseteq \exists hasChild$, restricting z, then

$$Ecert(q_c^k, \mathcal{T}_{fam}, \mathcal{D}_{fam}) = \{\langle \texttt{Luisa}, 100\rangle \langle \texttt{Anne}, 150\rangle\}$$

(ii) Otherwise

$$Ecert(q_c^k, \mathcal{T}_{fam}, \mathcal{D}_{fam}) = \emptyset$$

Restrictedness guarantees non-emptyness with "intensional" aggregates

Properties

THEO. (**count**)

THEO. (**countd**,**min**,**max**)

Properties

THEO. (**sum**)

THEO. (**avg**)

Concluding Remarks

Concluding Remarks

We have proposed a new class of queries: epistemic aggregate queries, and studied their syntax, semantics and (some of) their basic properties

Concluding Remarks

We have proposed a new class of queries: epistemic aggregate queries, and studied their syntax, semantics and (some of) their basic properties

EAQs allow us to define a meaningful notion of certain answers for SQL aggregation functions in an ODBAS setting

Concluding Remarks

We have proposed a new class of queries: epistemic aggregate queries, and studied their syntax, semantics and (some of) their basic properties

EAQs allow us to define a meaningful notion of certain answers for SQL aggregation functions in an ODBAS setting

We would like to consider more expressive ontology languages, allowing global and local arbitrary cardinality constraints.

Concluding Remarks

We have proposed a new class of queries: epistemic aggregate queries, and studied their syntax, semantics and (some of) their basic properties

EAQs allow us to define a meaningful notion of certain answers for SQL aggregation functions in an ODBAS setting

We would like to consider more expressive ontology languages, allowing global and local arbitrary cardinality constraints.

We know that answering EAQs is computable, but their complexity is still an open issue

Concluding Remarks

We have proposed a new class of queries: epistemic aggregate queries, and studied their syntax, semantics and (some of) their basic properties

EAQs allow us to define a meaningful notion of certain answers for SQL aggregation functions in an ODBAS setting

We would like to consider more expressive ontology languages, allowing global and local arbitrary cardinality constraints.

We know that answering EAQs is computable, but their complexity is still an open issue

We would also like to study the problems of containment and equivalence of EAQs (i.e. optimization)