Practical Update Management in Ontology-based Data Access

Giuseppe De Giacomo[1] Domenico Lembo[1] Xavier Oriol[2]
Domenico Fabio Savo[1] Ernest Teniente[2]

[1] Sapienza Università di Roma

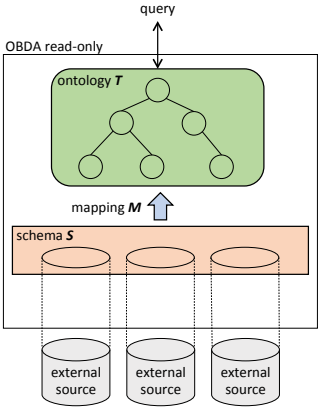
SAPIENZA
UNIVERSITÀ DI ROMA

[2] Universidad Politècnica de Catalunya

16th International Semantic Web Conference (ISWC 20017)
October 21th–25th, 2017, Vienna, Austria

(Read-only) Ontology-based Data Access Systems

Ontology-based data access (OBDA) is a (virtual) data integration paradigm.

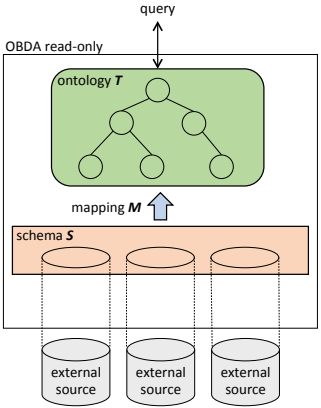


- The ontology provides a formal conceptualization of a domain of interest.
- The mapping specifies the relationship between the ontology and the source schema
- In OBDA, users access only the ontology, while the external data sources remain autonomous

– So far, huge research effort on query answering (**read-only OBDA**)
– Less attention to instance-level updates, i.e., changes at the extensional level of the OBDA system (write-also OBDA)

(Read-only) Ontology-based Data Access Systems

Ontology-based data access (OBDA) is a (virtual) data integration paradigm.

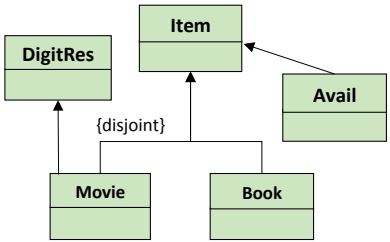


- The ontology provides a formal conceptualization of a domain of interest.
- The mapping specifies the relationship between the ontology and the source schema
- In OBDA, users access only the ontology, while the external data sources remain autonomous

– So far, huge research effort on query answering (**read-only OBDA**)
– Less attention to instance-level updates, i.e., changes at the extensional level of the OBDA system (write-also OBDA)

Example: ontology-level insertion

Ontology (TBox)



Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)

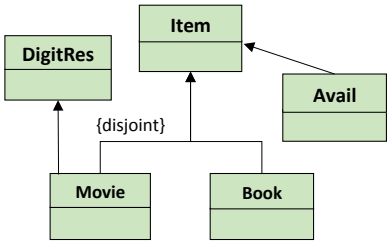
Mapping:
Movie(x):- T_Movie(x)
Book(x) :- T_Book(x)
Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z)

(Virtual) ABox:
Movie(Alien), Book(Ubik), Avail(Ubik)

Ontology-level update
Insert Item(Matrix) → If we push the update to the sources we have to insert T_Movie(Matrix), T_Book(Matrix), or T_Copy(Matrix,_?_)!

Example: ontology-level insertion

Ontology (TBox)



Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)

Mapping:
Movie(x):- T_Movie(x)
Book(x) :- T_Book(x)
Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z)

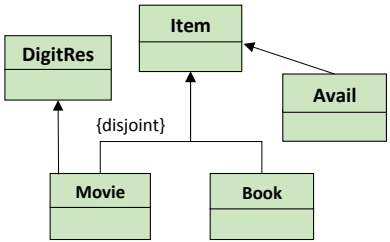
(Virtual) ABox:
Movie(Alien), Book(Ubik), Avail(Ubik)

Ontology-level update

Insert Item(Matrix) → If we push the update to the sources we have to insert T_Movie(Matrix), T_Book(Matrix), or T_Copy(Matrix,_?_)!

Example: ontology-level insertion

Ontology (TBox)



Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)

Mapping:
Movie(x):- T_Movie(x)
Book(x) :- T_Book(x)
Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z)

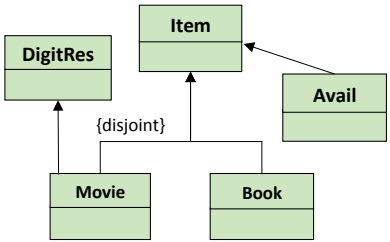
(Virtual) ABox:
Movie(Alien), Book(Ubik), Avail(Ubik)

Ontology-level update

Insert Item(Matrix) $\rightarrow$ If we push the update to the sources we have to insert T_Movie(Matrix), T_Book(Matrix), or T_Copy(Matrix,_?_)!

Example: ontology-level deletion

Ontology (TBox)



Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)

Mapping:
Movie(x):- T_Movie(x)
Book(x) :- T_Book(x)
Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z)

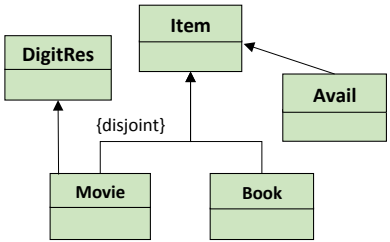
(Virtual) ABox:
Movie(Alien), Book(Ubik), Avail(Ubik)

Ontology-level update

Delete Avail(Ubik) → If we push the update to the sources, again we have to force unintended changes, e.g., dropping T_Copy(Ubik,C2), or adding T_Borrow(C2,_?_).

Example: ontology-level deletion

Ontology (TBox)



Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)

Mapping:
Movie(x):- T_Movie(x)
Book(x) :- T_Book(x)
Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z)

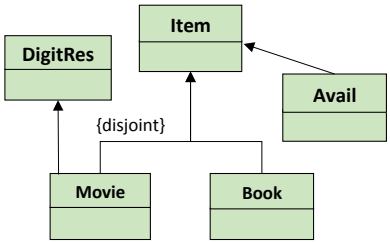
(Virtual) ABox:
Movie(Alien), Book(Ubik), Avail(Ubik)

Ontology-level update

Delete Avail(Ubik) → If we push the update to the sources, again we have to force unintended changes, e.g., dropping T_Copy(Ubik,C2), or adding T_Borrow(C2,_?_).

Example: ontology-level deletion

Ontology (TBox)



Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),
T_Borrow(C1,Bob)

Mapping:
Movie(x):- T_Movie(x)
Book(x) :- T_Book(x)
Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z)

(Virtual) ABox:
Movie(Alien), Book(Ubik), Avail(Ubik)

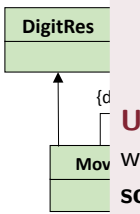
Ontology-level update

Delete Avail(Ubik) → If we push the update to the sources, again we have to force unintended changes, e.g., dropping T_Copy(Ubik,C2), or adding T_Borrow(C2,_?_).

Example: ontology-level deletion

Source Database:
T_Movie(Alien), T_Book(Ubik),
T_Copy(Ubik,C1), T_Copy(Ubik,C2),

Ontology (TBox)



Ontology-level Update (pushed to the sources)

Unwanted Behaviour: we cannot always find a reasonable way to push the update at the sources. Also, we **compromise sources autonomy**

ow(y,z)

l(Ubik)

Ontology-level update

Delete Avail(Ubik) → If we push the update to the sources, again we have to force unintended changes, e.g., dropping T_Copy(Ubik,C2), or adding T_Borrow(C2,_?_).

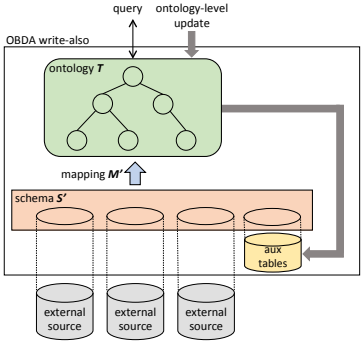
Write-also Ontology-based Data Access

We use auxiliary **ins/del tables** to store the changes without modifying the sources, and change the mapping $\mathcal{M}$ into a mapping $\mathcal{M}'$ in the following way:

1. For each atomic concept/role N, add $N(\vec{x})$:- $ins\_N(\vec{x})$;
2. Replace each $N(\vec{x})$:- $\phi(\vec{x})$, with $N(\vec{x})$:- $\phi(\vec{x}), \neg del\_N(\vec{x})$.

Mapping $\mathcal{M}'$:

Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z),
¬del_Avail(x)
Avail(x) :- ins_Avail(x)
Item(x) :- ins_Item(x)
...
...



⇒ To accomplish the updates of our example, we simply materialize ins_Item(Matrix) and del_Avail(Ubik)

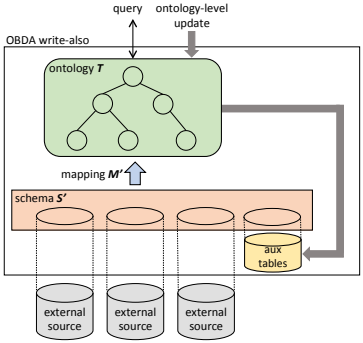
Write-also Ontology-based Data Access

We use auxiliary **ins/del tables** to store the changes without modifying the sources, and change the mapping $\mathcal{M}$ into a mapping $\mathcal{M}'$ in the following way:

1. For each atomic concept/role N, add $N(\vec{x})$:- $ins\_N(\vec{x})$;
2. Replace each $N(\vec{x})$:- $\phi(\vec{x})$, with $N(\vec{x})$:- $\phi(\vec{x}), \neg del\_N(\vec{x})$.

Mapping $\mathcal{M}'$:

Avail(x) :- T_Copy(x,y),¬T_Borrow(y,z),
¬del_Avail(x)
Avail(x) :- ins_Avail(x)
Item(x) :- ins_Item(x)
...
...



⇒ To accomplish the updates of our example, we simply materialize ins_Item(Matrix) and del_Avail(Ubik)

Dealing with inconsistency

Problem: new information may **contradict** the old one.

Example: We want **insert** Book(Alien), but the system implies Movie(Alien) and the ontology states that Book $\sqsubseteq \neg$Movie

We use an update operator that:

- obeys to the **success of update principle** (e.g., after the update Book(Alien) has to be **implied**);
- obeys to the **minimal change principle**: to preserve consistency **minimal** deletions are performed (e.g., Movie(Alien) has **not** to be implied after the update);
- performs additional **insertions to preserve** facts implied by the removed information if they do not contradict the update (e.g., since Movie $\sqsubseteq$ DigitRes, we insert DigitRes(Alien), that is a consequence of Movie(Alien) that is consistent with Book(Alien)).

This update semantics is presented in *[De Giacomo et al. ISWC16]* for stand-alone ontologies.
For *DL-Lite*$_\mathcal{A}$ (i.e., OWL 2 QL) ontologies the result of the update is **unique**, and can be computed through a non-recursive Datalog program.

Dealing with inconsistency

Problem: new information may **contradict** the old one.

Example: We want **insert** Book(Alien), but the system implies Movie(Alien) and the ontology states that Book $\sqsubseteq \neg$Movie

We use an update operator that:

- obeys to the **success of update principle** (e.g., after the update Book(Alien) has to be **implied**);
- obeys to the **minimal change principle**: to preserve consistency **minimal** deletions are performed (e.g., Movie(Alien) has **not** to be implied after the update);
- performs additional **insertions to preserve** facts implied by the removed information if they do not contradict the update (e.g., since Movie $\sqsubseteq$ DigitRes, we insert DigitRes(Alien), that is a consequence of Movie(Alien) that is consistent with Book(Alien)).

This update semantics is presented in *[De Giacomo et al. ISWC16]* for stand-alone ontologies. For *DL-Lite*$_{\mathcal{A}}$ (i.e., OWL 2 QL) ontologies the result of the update is **unique**, and can be computed through a non-recursive Datalog program.

Dealing with inconsistency

Problem: new information may **contradict** the old one.

Example: We want **insert** Book(Alien), but the system implies Movie(Alien) and the ontology states that Book $\sqsubseteq \neg$Movie

We use an update operator that:

- obeys to the **success of update principle** (e.g., after the update Book(Alien) has to be **implied**);
- obeys to the **minimal change principle**: to preserve consistency **minimal** deletions are performed (e.g., Movie(Alien) has **not** to be implied after the update);
- performs additional **insertions to preserve** facts implied by the removed information if they do not contradict the update (e.g., since Movie $\sqsubseteq$ DigitRes, we insert DigitRes(Alien), that is a consequence of Movie(Alien) that is consistent with Book(Alien)).

This update semantics is presented in *[De Giacomo et al. ISWC16]* for stand-alone ontologies.
For *DL-Lite*$_\mathcal{A}$ (i.e., OWL 2 QL) ontologies the result of the update is **unique**, and can be computed through a non-recursive Datalog program.

Non-recursive Datalog encoding for ontology-level updates

To compute the actions we need to accomplish ontology-level updates, we use a Datalog program (adapted from *[De Giacomo et al. ISWC16]*).

Example: As said, in response to an update **Insert** Book(Alien), our operator realizes the asked insertion, and additionally asks to **remove** Movie(Alien) and **insert** DigitRes(Alien).

ins_Book_req(Alien). T_Movie(Alien).

Book(x) :- T_Book(x). ins_Book'(x) :- ins_Book_req(x),¬Book(x).
Movie(x) :- T_Movie(x). del_Movie'(x) :- Movie(x), ins_Book_req(x).
ins_DigitRes'(x):- del_Movie'(x),¬del_DigRes(x).
... ...

According to the program results we modify ins/del tables as follows:

foreach *fact ins_N'($\vec{t}$)* **do if** *del_N($\vec{t}$)* **then** remove *del_N($\vec{t}$)* **else** insert *ins_N($\vec{t}$)*
foreach fact *del_N'($\vec{t}$)* **do if** *ins_N($\vec{t}$)* **then** remove *ins_N($\vec{t}$)* **else** insert *del_N($\vec{t}$)*

Non-recursive Datalog encoding for ontology-level updates

To compute the actions we need to accomplish ontology-level updates, we use a Datalog program (adapted from *[De Giacomo et al. ISWC16]*).

Example: As said, in response to an update **Insert** Book(Alien), our operator realizes the asked insertion, and additionally asks to **remove** Movie(Alien) and **insert** DigitRes(Alien).

ins_Book_req(A
Book(x)
Movie(x)
ins_DigitRes'(x
... ...

Theorem

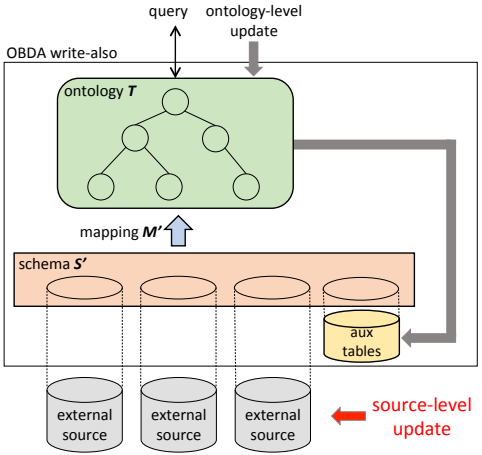
Ontology-level updates in OBDA systems with DL-Lite$_\mathcal{A}$ ontologies can be computed in $\mathbf{AC}^0$ *in data complexity.*

According to the program results we modify ins/del tables as follows:

foreach *fact ins_N'*$(\vec{t})$ **do if** *del_N*$(\vec{t})$ **then** remove *del_N*$(\vec{t})$ **else** insert *ins_N*$(\vec{t})$
foreach fact *del_N'*$(\vec{t})$ **do if** *ins_N*$(\vec{t})$ **then** remove *ins_N*$(\vec{t})$ **else** insert *del_N*$(\vec{t})$

Source-level updates

What happens when the sources are updated?



Source-level updates

What happens when the sources are updated?

- The sources are modified in any case.
- We compute how source updates **are reflected** at the ontology level, based on the mapping (we use event rules by *[Urpí et al. VLDB92]*).

  Example:

  $\text{ins\_Book\_sl(x)} \;\text{ :- ins\_T\_Book(x)}, \neg\, \text{T\_Book(x)}.$
  $\text{del\_Movie\_sl(x)} \text{ :- del\_T\_Movie(x)}, \text{ T\_Movie(x)}.$
  ...

  We align **ins/del** tables (e.g., remove ins_Movie(StarWars) if ins_Movie_sl(StarWars) holds because of ins_T_Movie(StarWars)) since this info comes now from the sources) .

  Also, we deal with inconsistencies as before (i.e., new insertions induced by source level updates are treated as ontology-level updates)

Source-level updates

What happens when the sources are updated?

- The sources are modified in any case.
- We compute how source updates **are reflected** at the ontology level, based on the mapping (we use event rules by *[Urpí et al. VLDB92]*).

Example:

$\text{ins\_Book\_sl(x)} \;\text{ :- ins\_T\_Book(x)}, \neg\, \text{T\_Book(x)}.$
$\text{del\_Movie\_sl(x)} \text{ :- del\_T\_Movie(x)}, \text{ T\_Movie(x)}.$
...

We align **ins/del** tables (e.g., remove ins_Movie(StarWars) if ins_Movie_sl(StarWars) holds because of ins_T_Movie(StarWars)) since this info comes now from the sources) .

Also, we deal with inconsistencies as before (i.e., new insertions induced by source level updates are treated as ontology-level updates)

Source-level updates

What happens when the sources are updated?

- The sources are modified in any case.
- We compute how source updates **are reflected** at the ontology level, based on the mapping (we use event rules by *[Urpí et al. VLDB92]*).

  Example:

  $\text{ins\_Book\_sl(x)} \;\text{ :- ins\_T\_Book(x)}, \neg\, \text{T\_Book(x)}.$
  $\text{del\_Movie\_sl(x)} \text{ :- del\_T\_Movie(x)}, \text{ T\_Movie(x)}.$
  ...

  We align **ins/del** tables (e.g., remove ins_Movie(StarWars) if ins_Movie_sl(StarWars) holds because of ins_T_Movie(StarWars)) since this info comes now from the sources) .

  Also, we deal with inconsistencies as before (i.e., new insertions induced by source level updates are treated as ontology-level updates)

Turning source-level into ontology-level updates: Incoherency problem

ontology-level updates induced by source-level ones may be incoherent (i.e., the update alone might contradict the ontology):

- As usual in update theory, we assume that ontology-level updates directly specified by users are coherent with the ontology.
- Instead, ontology-level updates induced by source level ones simply reflect the behaviour of the (autonomous) sources, which are not required to maintain data that satisfy the ontology.

Example: a source-level update may at the same time insert both

$$\mathsf{T\_Movie(DaVinciCode)} \text{ and } \mathsf{T\_Book(DaVinciCode)},$$

which turned into an ontology-level update actually asks to insert both

$$\mathsf{Movie(DaVinciCode)} \text{ and } \mathsf{Book(DaVinciCode)},$$

which thus contradict $\mathsf{Movie} \sqsubseteq \neg\mathsf{Book}$.

Dealing with incoherent updates

We repair the ontology-level updates **inferred** by source-level ones:

Given one such update $\mathcal{U} = \langle \mathcal{A}^+, \mathcal{A}^- \rangle$, where $\mathcal{A}^+$ are insertions and $\mathcal{A}^-$ are deletions, and a *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$, we take the intersection $\mathcal{A}^+_{rep}$ of all the maximal $\mathcal{T}$-consistent subsets of the $\mathcal{T}$-closure of $\mathcal{A}^+$.
Then the repaired update is $\mathcal{U}_{rep} = \langle \mathcal{A}^+_{rep}, \mathcal{A}^- \rangle$.

Note: the $\mathcal{T}$-closure of an ABox $\mathcal{A}$ is the set $\{\alpha \mid \alpha \text{ is a fact s.t. } \langle \mathcal{T}_p, \mathcal{A} \rangle \models \alpha\}$, where $\mathcal{T}_p$ is the set of all positive inclusions in $\mathcal{T}$.

Example:
$\mathcal{A}^+ = \{\text{Book(DaVinciCode), Movie(DaVinciCode)}\}$
$\mathcal{T}$-**closure of** $\mathcal{A}^+ = \{\text{Book(DaVinciCode), Movie(DaVinciCode),}$
$\text{Item(DaVinciCode), DigitRes(DaVinciCode)}\}$
maximal consistent subsets of the $\mathcal{T}$-closure of $\mathcal{A}^+$:
$\mathcal{A}_{r1} = \{\text{Book(DaVinciCode),Item(DaVinciCode),DigitRes(DaVinciCode)}\}$
$\mathcal{A}_{r2} = \{\text{Movie(DaVinciCode),Item(DaVinciCode),DigitRes(DaVinciCode)}\}$
$\mathcal{A}^+_{rep} = \{\textbf{Item(DaVinciCode),DigitRes(DaVinciCode)}\}$

Dealing with incoherent updates

We repair the ontology-level updates **inferred** by source-level ones:

Given one such update $\mathcal{U} = \langle \mathcal{A}^+, \mathcal{A}^- \rangle$, where $\mathcal{A}^+$ are insertions and $\mathcal{A}^-$ are deletions, and a *DL-Lite*$_\mathcal{A}$ TBox $\mathcal{T}$, we take the intersection $\mathcal{A}^+_{rep}$ of all the maximal $\mathcal{T}$-consistent subsets of the $\mathcal{T}$-closure of $\mathcal{A}^+$.
Then the repaired update is $\mathcal{U}_{rep} = \langle \mathcal{A}^+_{rep}, \mathcal{A}^- \rangle$.

Note: the $\mathcal{T}$-closure of an ABox $\mathcal{A}$ is the set $\{\alpha \mid \alpha \text{ is a fact s.t. } \langle \mathcal{T}_p, \mathcal{A} \rangle \models \alpha\}$, where $\mathcal{T}_p$ is the set of all positive inclusions in $\mathcal{T}$.

Example:

$\mathcal{A}^+ = \{\text{Book(DaVinciCode), Movie(DaVinciCode)}\}$

$\mathcal{T}$-**closure of** $\mathcal{A}^+ = \{\text{Book(DaVinciCode), Movie(DaVinciCode),}$
$\text{Item(DaVinciCode), DigitRes(DaVinciCode)}\}$

maximal consistent subsets of the $\mathcal{T}$-closure of $\mathcal{A}^+$:

$\mathcal{A}_{r1} = \{\text{Book(DaVinciCode),Item(DaVinciCode),DigitRes(DaVinciCode)}\}$

$\mathcal{A}_{r2} = \{\text{Movie(DaVinciCode),Item(DaVinciCode),DigitRes(DaVinciCode)}\}$

$\mathcal{A}^+_{rep} = \{\textbf{Item(DaVinciCode),DigitRes(DaVinciCode)}\}$

Source-level updates: wrapping up the technique

1. We defined a second non-recursive Datalog program that
   - turns source-level updates into ontology-level ones (by adapting event rules from [Urpí et al. VLDB92])
   - separates coherent updates from incoherent ones
   - repairs incoherent updates
2. On the basis of the evaluation of the above program on top of the current state of the data at the sources and auxiliary tables and the update at the sources we
   (i) align ins/del tables, and
   (ii) perform coherent and repaired ontology-level updates through our ontology-level update procedure.

Theorem

Source-level updates in OBDA systems with DL-Lite$_\mathcal{A}$ ontologies can be computed in AC^0 in data complexity.

Source-level updates: wrapping up the technique

1. We defined a second non-recursive Datalog program that
   - turns source-level updates into ontology-level ones (by adapting event rules from [Urpí et al. VLDB92])
   - separates coherent updates from incoherent ones
   - repairs incoherent updates
2. On the basis of the evaluation of the above program on top of the current state of the data at the sources and auxiliary tables and the update at the sources we
   (i) align ins/del tables, and
   (ii) perform coherent and repaired ontology-level updates through our ontology-level update procedure.

Theorem

Source-level updates in OBDA systems with DL-Lite$_\mathcal{A}$ ontologies can be computed in AC^0 in data complexity.

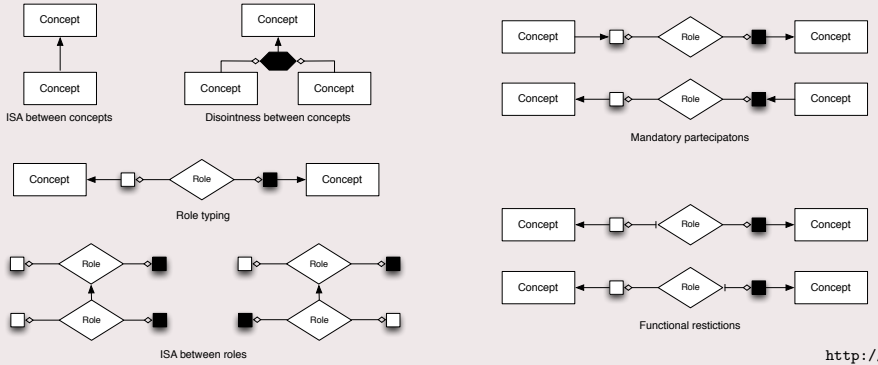
Conclusions

- We have defined a framework for write-also OBDA that allows for both ontology-level and sources-level updates.
- We also provide practical algorithms based on non-recursive Datalog programs for computing both kinds of updates.
- In doing this, we have devised solutions to deal with incoherent updates, as those possibly induced by source-level updates.
- We have shown that computing both ontology-level and source-level updates is in AC^0 w.r.t. data complexity, which is the usual desired complexity for OBDA tasks.

- We are currently working on an implementation and evaluation of our technique, as well as on the investigation of alternative semantics to deal with incoherent updates.


ngiyabonga
danke
謝謝
рахмат
teşekkür ederim
Баярлалаа
спасибо
thank you
gracias
tapadh leat
dank je
mochchakkeram
dziękuje
go raibh maith agat
obrigado
sukriya
kop khun krap
arigato
dakujem
merci
terima kasih
grazie
мерси
감사합니다
ধন্যবাদ
ευχαριστώ

DL-Lite$_\mathcal{A}$ constructs



Graphol diagrams
http://www.dis.uniroma1.it/~graphol

DL-Lite$_\mathcal{A}$ key properties

- Most expressive variant of *DL-Lite*
- Essentially OWL 2 QL + functional restrictions
- Captures most UML class diagrams/ER (except complete hierarchies)
- A bit first-order rewritability of UCQ