Datalog Extensions as Ontology Representation Languages and their Applications

Leopoldo Bertossi*

Carleton University

School of Computer Science

Ottawa, Canada

$\star$: Faculty Fellow of the IBM Center for Advanced Studies

A Start: Metadata in Data Management

- Metadata (MD) is data about data

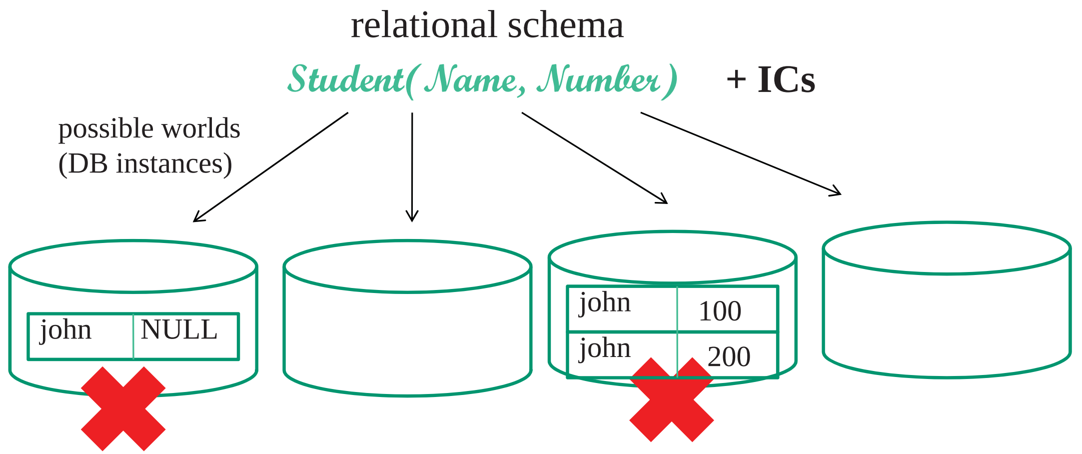
  An upper layer that gives information about a lower layer

  For example, about the data in relational tables

- We already know about MD in relational DBs: schemas, data types and domains, integrity constraints (ICs)

- If ICs are satisfied by the DB (as expected, but not always true), they provide synthetic, higher-level knowledge
  - ICs capture semantics (meaning) of data [3, 11]
  - By filtering out inadmissible (inconsistent) instances, the spectrum of possible instances is narrowed down

    By doing so, better targeting the intended meaning
  - Decreasing uncertainty



+ $\forall xy (Student(x, y) \rightarrow y \neq \text{NULL})$ **+** $\forall xyz (Student(x, y) \wedge Student(x, z) \rightarrow y = z)$

(capturing semantics via ICs, eliminating possible worlds)

- ICs tell us something about the stored data, still not much though
- ICs can be used, e.g. at query answering time

  For semantic query optimization
- ICs are also useful for interoperability purposes

  When data systems have to interact and possibly be integrated

  They tell us something about what's stored in the data source

Why not going beyond in terms of MD?

What else do we know or have after having created a relational DB?

Recovering ER models as Metadata

- When creating a database, we usually start from en entity-relationship (ER) model
- An ER model represents an external, data-related reality

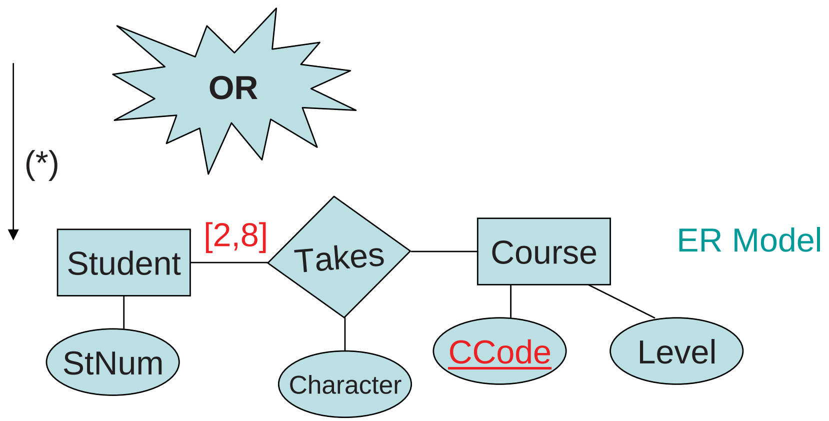
  For example, a model of a business environment

  The model is given as an ER diagram (UML diagram)
- The ER model is closer to the reality than the relational DB to be (which is also a model)
- The ER model is usually forgotten after the DB is created
- The ER model could be used as metadata!

- When creating a relational database, we usually start from an outside reality (OR), e.g. a company, a university, etc.

  We want to model that OR, i.e. produce an abstract, simplified description or representation of OR (leaving aside non-relevant, contingent aspects and details)

- A model can be an ER model, in terms of entities and relationships between them



- For the model to be a good model of OR, it must have a semantics or meaning that corresponds to OR

  ... and keeps the correspondence (*) in place (semantically correct)

- That is why we impose in the model some semantic constraints, like those in red in it

  - A student must take between 2 and 8 courses

  - The course code is a key for the entity: If two objects in *Course* coincide in their values for *CCode*, then their other attribute values must coincide too

- Without those constraints, there could be too many possible ORs that conform to the ER model

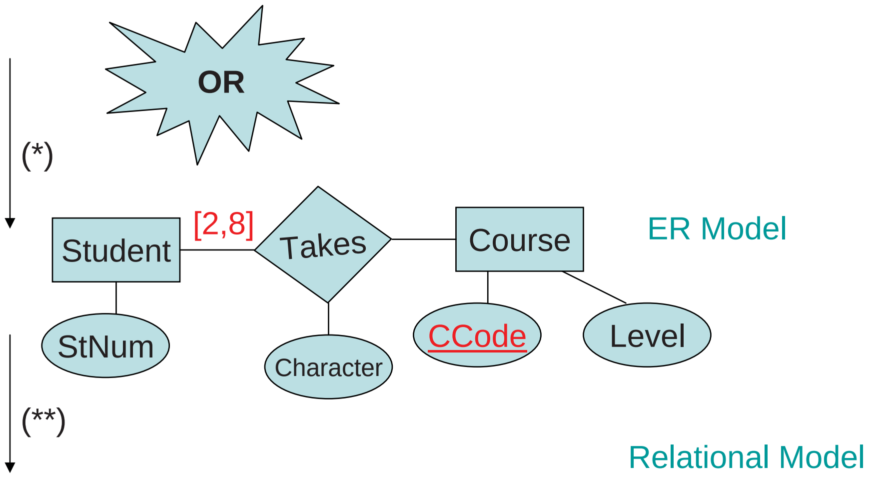
  The model becomes too ambiguous or uncertain

- Imposing semantic constraints eliminates unintended ORs

  ... by narrowing down meaning and filtering out undesirable ORs (other than the intended one)

  We want the ER model to be as close as possible to the initial OR

- The usual next step is producing a relational model from the ER model



Relational Schema: Student(StNum,Address), Course(CCode,Level,MaxReg), Takes(StNum,CCode,Character) + ICs

- The relational model is also a model of OR

- Now a logical model that uses the languages of predicate logic and set theory

- The relational ICs become part of the model, and are also semantic constraints

  Some of them come from the original ER model with its semantics constraints

- As mentioned above, the ER model may be discarded (or not used) after the relational DB is created and populated

  But the ER model contains much semantic information

  It could be put to good use: It could become metadata

  A semantic layer -that can be used with the DB- and is closer to OR and what the user understands

How to combine a diagrammatic model with a logical model?

How to realize the integration?

So that a computer system can take advantage of the combination ...

We could borrow languages that have been designed for- or applied to the Semantic Web (SW) initiative [2, 16, 12]

Some of those languages are being used to express ontologies as metadata for data sources

ER Models as Ontologies and OBDA

Logical languages to express metadata can interact with the logical data model (database)

Being the ER model a diagrammatic model, it can be reconstructed as a symbolic and logic-based ontology

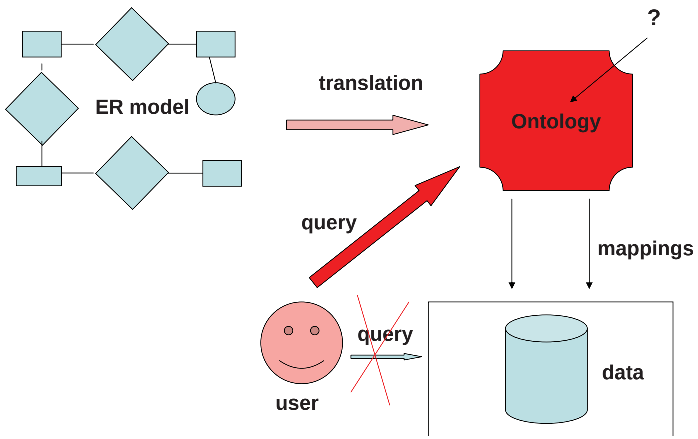
In general, an ontology is a (logical) description of a set of concepts and their relationships [9]

The ontology becomes metadata, now an explicit and formal ER model

The ontology (ex ER model) -being closer to the user or business reality- can be used to query the DB

Querying data sources through ontologies is an active research area

OBDA: Ontology-based data access [14]



ER model is replaced by (reconstructed as) a symbolic, logical *ontology*

For example, for the following entities/relationship

Introduce basic predicates for the ontology:

- Unary predicates for concepts: $Employee(\cdot)$
- Binary predicates for roles: $BossOf(\cdot,\cdot)$, $ReportsTo(\cdot,\cdot)$

Symbolic statements go into the ontology

E.g. to capture the $(0,1)$ constraint on the ER's **reportTo**: *"Every employee reports to at most one employee"* :

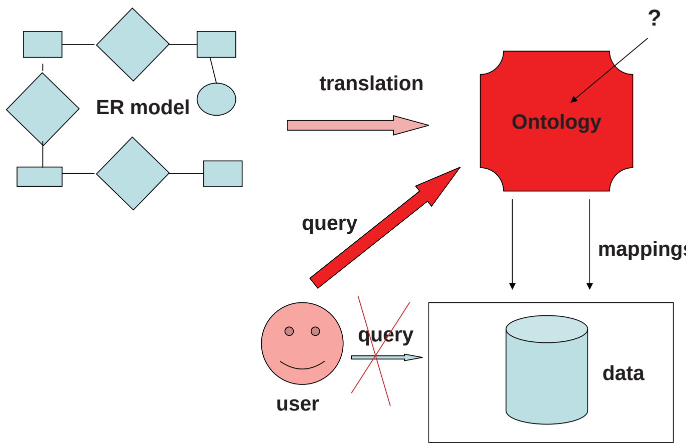
$\forall x(Employee(x) \rightarrow \exists^{\leq 1} y(Employee(y) \wedge ReportsTo(x,y))$[1]

A symbolic, machine-processable sentence ...

Back to OBDA ...

Query language is the language of the ontology

Data stay underneath



Ontology queries are internally "translated" into DB queries

For that, use the mappings between the ontology and the underlying database (data source)

[1] I.e., $\forall x(Employee(x) \rightarrow \forall x \forall y_1 y_2((Employee(y_1) \wedge ReportsTo(x,y_1) \wedge Employee(y_2) \wedge ReportsTo(x,y_2)) \rightarrow y_1 = y_2)$. If the ER constraint were $(1,1)$, it would be: $\forall x(Employee(x) \rightarrow \exists y(Employee(y) \wedge ReportsTo(x,y)) \wedge \forall x \forall y_1 y_2((Employee(y_1) \wedge ReportsTo(x,y_1) \wedge Employee(y_2) \wedge ReportsTo(x,y_2)) \rightarrow y_1 = y_2)$

Just for the gist:

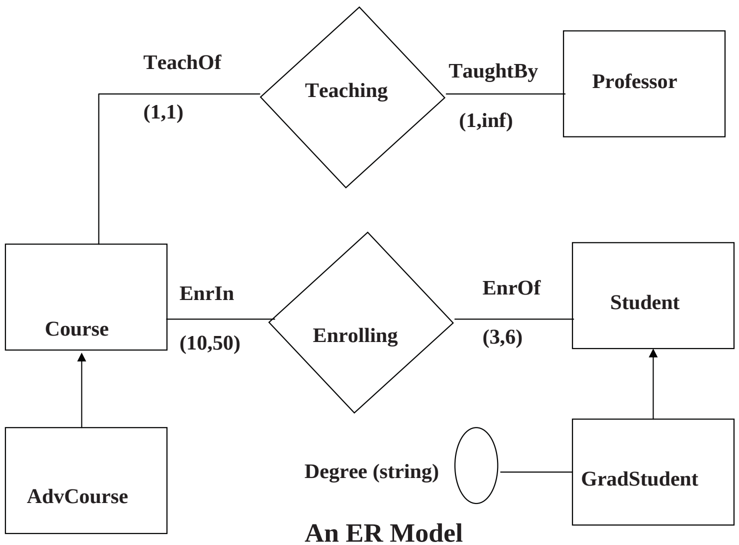
The link between **AdvCourse** and **Course** is an IS-A link

As an ontology written in Description Logic (DL)

Entities become DL-concepts; ER links become DL-roles (binary predicates)

DL is at the basis of SW languages, such as OWL

($\sqsubseteq$ is $\subseteq$ or $\longrightarrow$; $\sqcap$ is $\cap$ or $\wedge$; $^-$ denotes the inverse role (predicate); original constraints in red)



An ER Model

$$\begin{array}{rcl}
Teaching & \sqsubseteq & \forall TeachOf.Course \ \sqcap \ \exists^{=1} TeachOf \ \sqcap \\
 & & \forall TaughtBy.Professor \ \sqcap \ \exists^{=1} TaughtBy \\
Enrolling & \sqsubseteq & \forall EnrIn.Course \ \sqcap \ \exists^{=1} EnrIn \ \sqcap \\
 & & \forall EnrOf.Student \ \sqcap \ \exists^{=1} EnrOf \\
Course & \sqsubseteq & \forall TeachOf^{-}.Teaching \ \sqcap \ \exists^{=1} TeachOf^{-} \ \sqcap \\
 & & \forall EnrIn^{-}.Enrolling \ \sqcap \ \exists^{\geq 10} EnrIn^{-} \ \sqcap \ \exists^{\leq 50} EnrIn^{-} \\
AdvCourse & \sqsubseteq & Course \\
Professor & \sqsubseteq & \forall TaughtBy^{-}.Teaching \\
Student & \sqsubseteq & \forall EnrOf^{-}.Enrolling \ \sqcap \ \exists^{\geq 3} EnrOf^{-} \ \sqcap \ \exists^{\leq 6} EnrOf^{-} \\
GradStudent & \sqsubseteq & Student \ \sqcap \ \forall Degree.String \ \sqcap \ \exists^{=1} Degree
\end{array}$$

The mappings are between unary and binary predicates in the ontology and database predicates (tables), which can be of any arity

The restricted syntax of DL makes automated reasoning feasible, and sometimes, also efficient

Notice that full classical predicate logic of which (most of the variants of) DL is a (are) fragment(s) is provably undecidable

The DL ontology above could be written in OWL

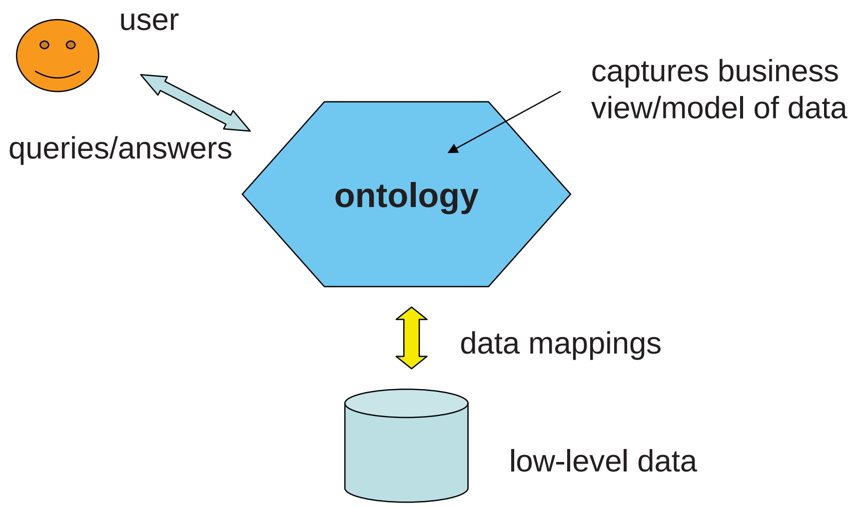
(Above, ER constraints captured in red in the ontology)

By reasoning we can infer that constraints that apply to $Course$ also apply to $AdvCourse$

And less direct logical consequences from the ontology

Ontologies can be more expressive than ER models

We could start directly with/from an ontology (not necessary coming from an ER model)



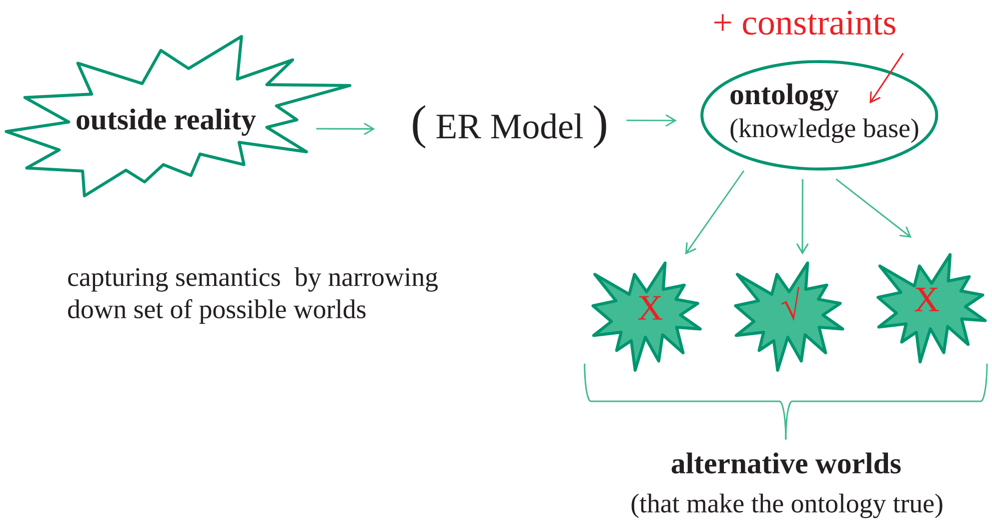
Their logic-based languages have precise syntax and semantics

The ontology can be used to capture more semantics

... in declarative, precise, and executable terms ...

It is possible to do automated reasoning from those ontologies

Via extra logical conditions (constraints) unintended possible worlds that make the ontology true (satisfy the ontology) can be filtered out (cf. page 3)



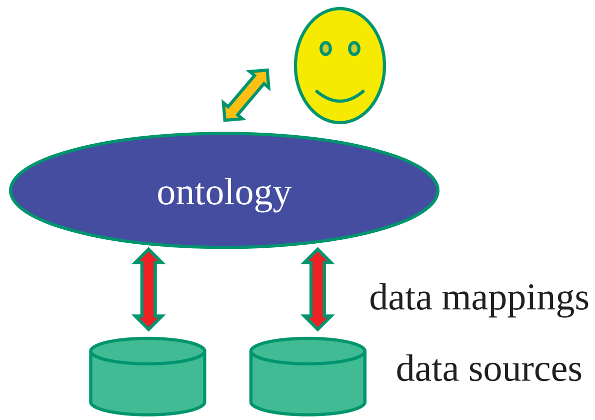
This ontology-based approach enables conceptually simpler and more flexible integration of data management with higher-level reasoning systems

Those ontologies can be useful for interoperability and integration purposes [15]

The Virtual Data Integration Connection

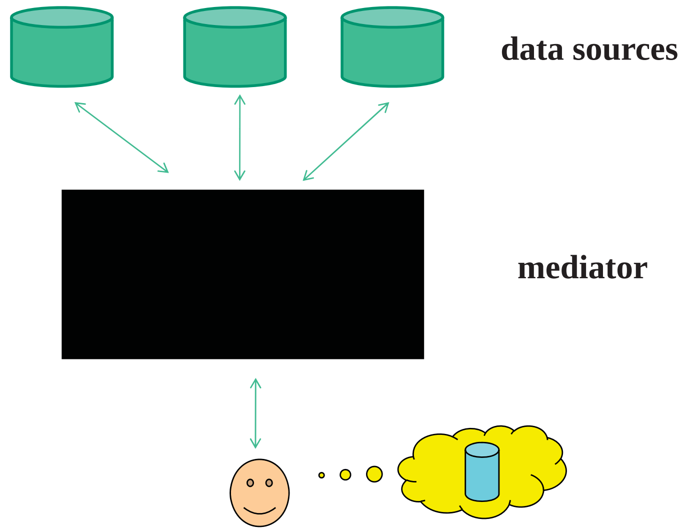
Other data sources could be added under an ontology

Integrating data sources through the same ontology



Data source integration is a crucial problem in business applications, bioinformatics, etc.

A classical virtual approach to data integration is via a mediator [17]



SW system offering DB-like schema interface

User queries the mediator, and data stay at the sources

Mappings allow meadiator to send ad hoc queries to sources

Example: Want to virtually integrate CU and OU DBs

Sources: Carleton U. Ottawa U.

CUstudents	Number	Name
	101	john
	102	mary

OUstudents	Number	Name
	103	claire
	101	peter

SpecialCU	Number	Field
	101	alg
	102	ai

SpecialOU	Number	Field
	101	db

Single global relation schema at mediator:

$$Students(Number, Name, Univ, Field)$$

User queries in terms of $Students$

Mappings between the source schemas and the mediated schema?

The mappings above are stored at- and managed by the mediator

The logical part (the non-procedural components) of the mediator could be conceived as an ontology

A logical schema mapping: (uses two Datalog rules for view definitions)

$$CUstudents(x,y), SpecialCU(x,z) \rightarrow Students(x,y,\text{'cu'},z)$$

$$OUstudents(x,y), SpecialOU(x,z) \rightarrow Students(x,y,\text{'ou'},z)$$

Students becomes a view defined as a disjunction of two conjunctive queries

Global relation as a view of source relations (not the only possibility)

(Can be put as a view defined in relational calculus:

$$\forall xyuz[(CUstudents(x,y) \wedge SpecialCU(x,z) \wedge u = \text{'cu'}) \vee$$
$$(OUstudents(x,y) \wedge SpecialOU(x,z) \wedge u = \text{'ou'}) \rightarrow Students(x,y,u,z)]$$

What Languages for ODBA?

- We saw that dialects of DL could be such ontological languages
- Something closer to database practice?
- Datalog has been around for some years in the DB community

  As a query and view definition language for relational DBs

  As opposed to relational algebra/calculus and older versions of SQL, Datalog provides recursion

Parent	A1	A2
	juan	pablo
	adam	cain
	adam	abel
	eve	cain
	pablo	luis

$$Ancestor(x, y) \leftarrow Parent(x, y)$$
$$Ancestor(x, z) \leftarrow Ancestor(x, y), Parent(y, z)$$

- Datalog has many nice properties and implementations, but also limited expressive power

- Can we extend Datalog to make it more expressive while keeping most of its nice properties?

Datalog$\pm$ as an Ontological Framework

- Datalog$\pm$ is a family of extensions of classic Datalog, with new kinds of rules and constraints [5, 8]
- Its languages allow to represent ontological axioms and integrity constraints that cannot be expressed in Datalog
- The idea is to extend Datalog with new constructs to gain expressive power
- While trying to keep the good properties of Datalog:

$\longrightarrow$ declarativity, clear logical semantics,
effectiveness & efficiency

(as extensions of whatever available for Datalog)

Most prominent new ingredients: (the "$+$" in Datalog$\pm$)

- Rules in Datalog$\pm$ admit existentially quantified variables:

$$\exists x P(x,y) \leftarrow R(y,z)$$

Can be seen as tuple-generating dependencies (TGDs)

- Negative Constraints (NCs): (in particular, denial constraints)

$$\perp \leftarrow P(x,y), R(y,z)$$

- Equality generating dependencies (EGDs):

$$y = z \leftarrow P(x,y), P(x,z)$$

In this case, a key constraint (KC)

Example: An incomplete EDB D of employers and employees

- Impose on D the TGD (usually as an inclusion dependency):

  "every manager is an employee"

  Expressed by a Datalog rule: $employee(x) \leftarrow manager(x)$

- Another TGD: *"every manager supervises someone"*

  As a rule in Datalog$\pm$: $\exists y\ supervises(x, y) \leftarrow manager(x)$

- Impose IC: *"employees are not employers"*

  As negative constraint (NC): $\perp \leftarrow employee(x), employer(x)$

- An EGD: *"every employee is supervised by at most one manager"*

$$x = x' \leftarrow supervises(x, y), supervises(x', y)$$

Several applications:

- Express/represent ontologies that interact with data sources
- Represent conceptual data models, and semantic layers on top of databases
- Datalog$\pm$ ontologies can represent: ER [7], Semantic Web languages/ontologies [4, 1], UML with object classes [6], ...
  (but not possible in classical Datalog!)
- Ontology-Based Data Access (OBDA)
  - Query a database through the ontology
  - In the language of the ontology (better understood by- and closer to the user)
  - Automatically access the underlying data sources
  - Get answers through Datalog evaluation

- Representation of- and navigation in multidimensional data models for data quality assessment and cleaning [13]

Properties & issues:

- The "$-$" in Datalog$\pm$ refers to syntactic restrictions we impose on the rules and their (syntactic) interactions
- This limits the gained expressive power
- We can still use Datalog$\pm$ to express ER models and much more
- It can be used as an ontological language
- It captures and extends the expressive power of light-weight DLs used for OBDA

- The mappings are part of the program

  They do not have to point to unary/binary ontological predicates only

  Predicates of arbitrary arity at the ontological level

- Seamless integration of source and ontological predicates in the Datalog$\pm$ ontology

- Datalog$\pm$ can be used as a language to extend incomplete DBs

  E.g. in page 25 we may have only extensional data for $manager$

- The syntactic restrictions ensure that query evaluation (QE) becomes feasible and sometimes efficient

  (Without them, QE under Datalog$\pm$ can be undecidable/non-computable)

- Datalog$\pm$ is still declarative and has a precise and clean semantics

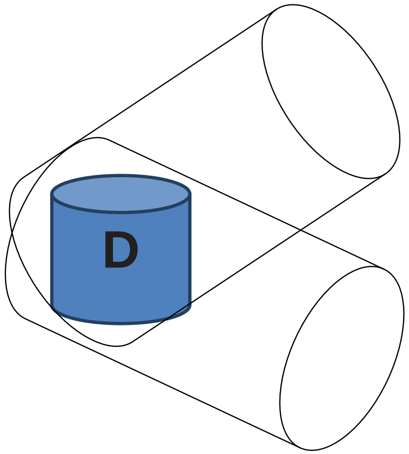
- QE can be implemented

Towards Good Members of the Datalog± Family

- A Datalog± program with a new kind of rules and classical ones is combined with an extensional database (EDB)
- EDB is considered to be incomplete, but extended through the Datalog± programs

  Generating new tuples for EDB predicates and full extensions for intensional predicates
- Depending on the kind of rules, possibly several extensions
- We may want to materialize the extension(s) or keep them virtual

  And query them ...



extensions via rules

Extensions are DBs that extend the EDB and satisfy the rules as classical logical formulas

Whatever is *true in all extensions*, i.e. *certain*

The chase (of the rules on the EDB) generates an instance that extends the EDB and “represents” the whole class of extensions

It turns out that what is certain is what is true in (the extension produced by) the chase

Example: Incomplete EDB $D = \{person(John)\}$

TGDs applied forward (as usual in Datalog), with value invention for existentials

This is the main part of the "chase procedure"

Set Σ of Datalog[$\exists$] rules:

$$\exists x \; father(x, y) \leftarrow person(y)$$
$$person(x) \leftarrow father(x, y)$$

The chase is a procedure that applies the TGDs in a forward manner, generating new tuples

$$\begin{aligned} chase(D, \Sigma) = \{ & father(z_1, John), person(z_1), \\ & father(z_2, z_1), person(z_2), \\ & father(z_3, z_2), person(z_3), \ldots\} \end{aligned}$$

(each z_i is a labeled null value)

- Chase may create non-terminating loops

  So, the chase may not terminate

  Query answering may become undecidable

- Related to (but not necessarily implied by) the fact that ...

  The chase procedure for Datalog[$\exists$] may not terminate, i.e. it produces an infinite extension

  Finite or infinite, we may still query it ...

- Query answering under Datalog[$\exists$] is indeed undecidable

- Even with infinite chase, things are not always hopeless ...

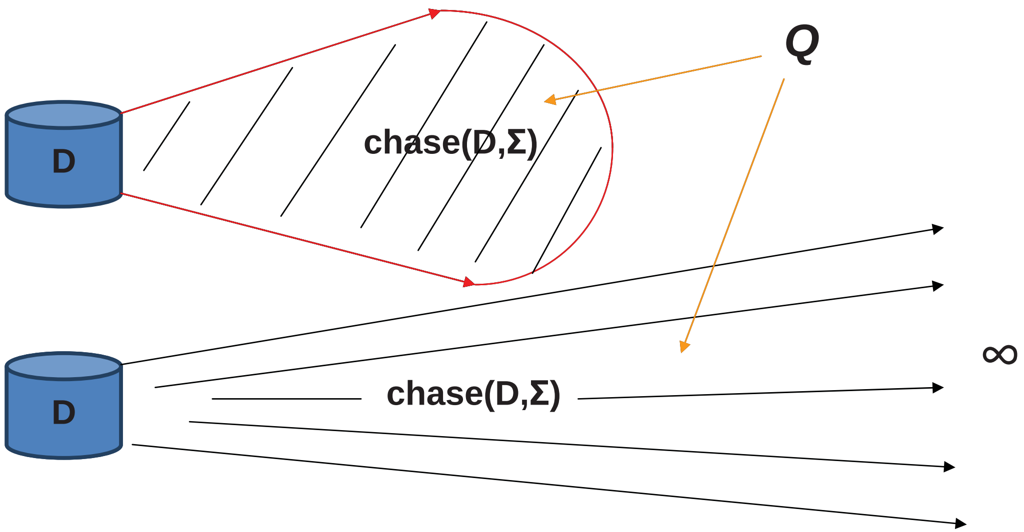
- Idea: impose syntactic restrictions of Datalog$\pm$ programs

  To guarantee decidability of query answering

  And hopefully efficient query answering ...

- We may reserve the term Datalog$\pm$ for the “good” extensions of Datalog

  Each of them (at least the TGD part) can be seen as a syntactic fragment of Datalog[$\exists$] (the extension of Datalog with unrestricted existential rules)



- In first case, QA is obviously decidable

  If the chase can be built in PTIME (in data), QA too

- In second case, QA may be (and sometimes is) undecidable

  But also possibly decidable depending on the program (and the class of queries, but we assume them conjunctive)

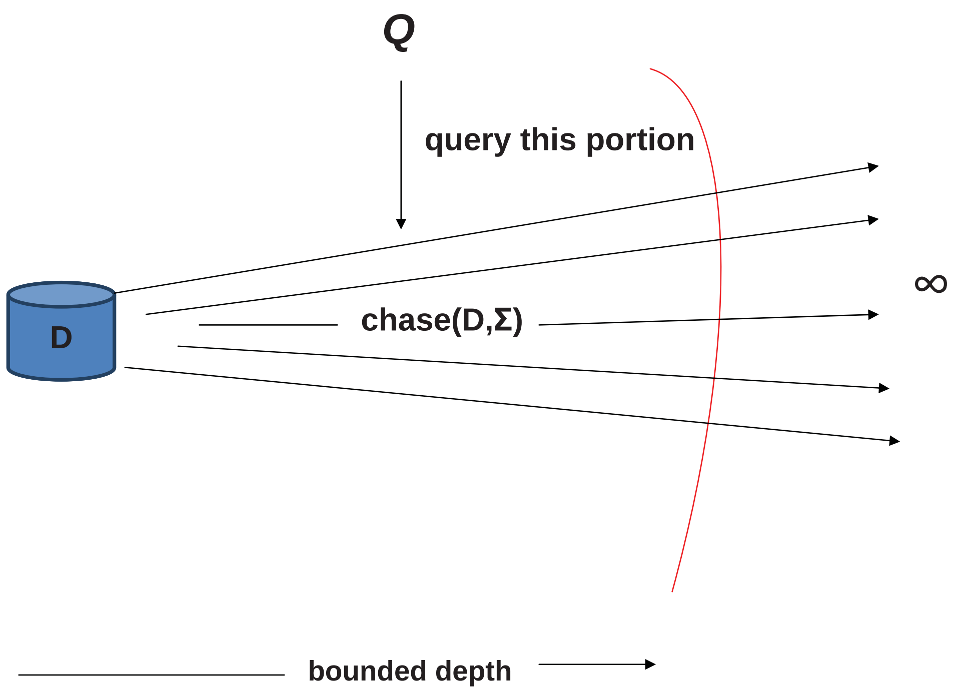
- Good cases of programs that ensure decidability of QA?

  And efficient QA?

- Well-behaved classes of Datalog[∃] programs have been considered for the second (infinite) case

- Decidability of QA guaranteed by different syntactic conditions on the set of rules

- The idea is that, depending on the programs, QA can be correctly done by querying only a bounded, initial portion of the chase

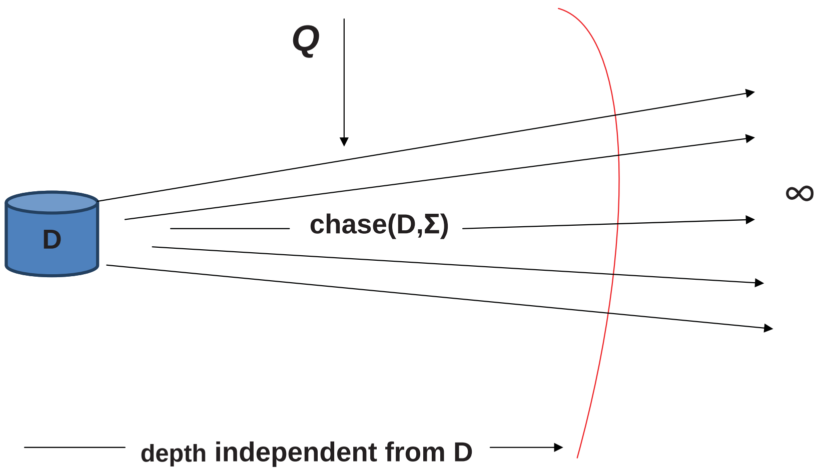


Hopefully a "short portion"

Two good cases:

(A) Bound independent from D (but dependent on $\mathcal{Q}, \Sigma$):

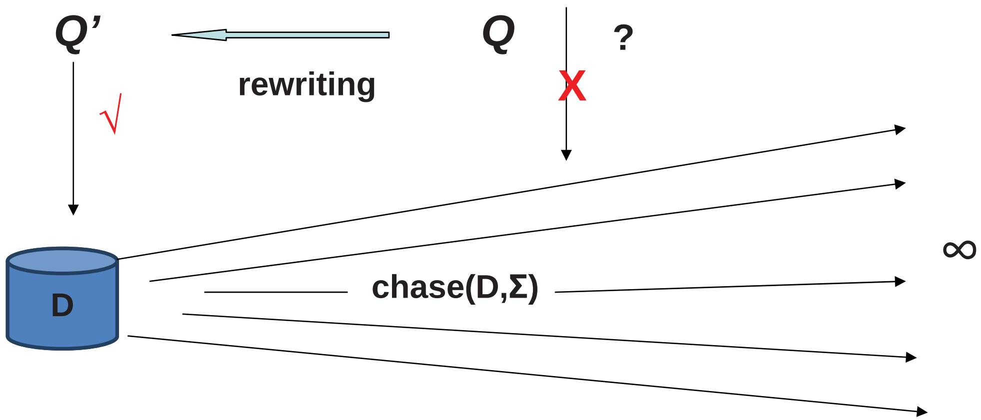
BDDP: bounded derivation depth property



- In this case, FO query rewriting is possible (more on this below)

  Rewriting via rules in the program

- Instead of posing the query to the (infinite) chase, rewrite the query $\mathcal{Q}$ into a new FO query $\mathcal{Q}'$ (independently from D)
- Query D with $\mathcal{Q}'$ as usual
- Definitely in PTIME in data



(B) Bound depends polynomially on (size of) D

- (A) is a particular case of (B)
- To achieve (B) (or (A)), different syntactic restrictions on Datalog[$\exists$] programs
- Identified various classes of Datalog$\pm$ programs: linear, guarded, sticky, weakly-sticky, ...
- For some, even (A) is possible, e.g. sticky Datalog$\pm$ programs

Sticky Datalog±

- Sticky programs enjoy BDDP (and then also FO-rewriting)
- The chase has stickiness property, which syntactically depends on the whole program

Example: A program with the stickiness property

$$\exists w\ T(x,y,w) \leftarrow R(x,y), P(y,z)$$

$$\exists w\ S(y,w) \leftarrow T(x,y,z)$$

(repeated/join variables in a body stick in the chase)

A non- sticky program:

$$\exists w\ T(x,y,w) \leftarrow R(x,y), P(y,z)$$

$$\exists w\ S(x,w) \leftarrow T(x,y,z)$$

- In sticky programs, join variables recursively stick to their consequences

- Stickiness can be checked syntactically, through a two-step marking procedure on a set of TGDs Σ

Example:

$$\begin{array}{rcl} \exists x, y, z \; emp(w, x, y, z) & \leftarrow & dept(v, w) \\ \exists z \; dept(w, z), runs(w, y), in\_area(y, x) & \leftarrow & emp(v, w, x, y) \\ \exists z \; external(z, y, x) & \leftarrow & runs(w, x), in\_area(x, y) \end{array}$$

1. Preliminary step: For each $\sigma \in \Sigma$ and variable $x \in body(\sigma)$, if there is an atom $a \in head(\sigma)$ such that x does not appear in a, mark each occurrence of x in $body(\sigma)$

$$\begin{array}{rcl} \exists x, y, z \; emp(w, x, y, z) & \leftarrow & dept(v, w) \\ \exists z \; dept(w, z), runs(w, y), in\_area(y, x) & \leftarrow & emp(v, w, x, y) \\ \exists z \; external(z, y, x) & \leftarrow & runs(w, x), in\_area(x, y) \end{array}$$

$$
\begin{aligned}
\sigma_1\text{: } \exists x, y, z \; emp(w, x, y, z) &\quad \leftarrow \quad dept(v, w) \\
\sigma_2\text{: } \exists z \; dept(w, z), runs(w, y), in\_area(y, x) &\quad \leftarrow \quad emp(v, w, x, y) \\
\sigma_3\text{: } \exists z \; external(z, y, x) &\quad \leftarrow \quad runs(w, x), in\_area(x, y)
\end{aligned}
$$

2. Propagation step (until fixed point reached): For each $\sigma \in \Sigma$, if a marked variable in $body(\sigma)$ appears at position p, then for every $\sigma' \in \Sigma$ (including σ), mark each occurrence of the variables in $body(\sigma')$ that appear in $head(\sigma')$ in same position p

$$
\begin{array}{lllll}
\sigma_2 : & & \leftarrow & emp[1] & : v \\
\sigma_1 : & emp[1] & \leftarrow & & : w \\
\sigma_1 : & & \leftarrow & dept[2] & : w
\end{array}
$$

$$
\begin{aligned}
\exists x, y, z \; emp(w, x, y, z) &\leftarrow dept(v, \underline{w}) \\
\exists z \; dept(w, z), runs(w, y), in\_area(y, x) &\leftarrow emp(v, w, x, y) \\
\exists z \; external(z, y, x) &\leftarrow runs(w, x), in\_area(x, y)
\end{aligned}
$$

- Σ is sticky if no marked variable appears more than once in a $body(\sigma)$

This one is!

Revisiting FO Rewriting-Based QA in Datalog$^\pm$

- Given: EDB D, a set of TGDs Σ (with BDDP), and conjunctive query $\mathcal{Q}$ (NCs and EGDs not considered here; see below)
- Construct FO rewriting $\mathcal{Q}_R$ of $\mathcal{Q}$ via Σ

  It holds: $\mathcal{Q}_R(D) = ans(\mathcal{Q}, D, \Sigma)$ (the certain answers)
- Evaluate $\mathcal{Q}_R$ over D
- All this as long as:
  - NCs hold, which can be checked separately by running associated conjunctive queries
  - EGDs do not interact with TGDs (during the chase), a separability property, which can be syntactically checked

- A rewriting algorithm is proposed for Datalog$\pm$ programs based on the iteration of two steps: [10]

  - Basic rewriting using the rules (resolution)
  - Minimization of the query obtained from rewriting step

  $\mathcal{Q}_R$ is the union of resulting conjunctive queries from iterations of the above

Example: (ex. in page 39 cont.) CQ $\mathcal{Q}$

$$q(p) \leftarrow \mathit{in\_area}(p,a),\ \mathit{external}(e,a,p)$$

Applying the TGD:

$$\exists z\ \mathit{external}(z,y,x) \leftarrow \mathit{runs}(w,x),\ \mathit{in\_area}(x,y)$$

basic rewriting step returns new CQ $\mathcal{Q}_1$: ($*$: don't care symbol)

$$q(p) \leftarrow \underline{\mathit{in\_area}(p,a)},\ \mathit{runs}(*,p),\ \underline{\mathit{in\_area}(p,a)}$$

Minimization leads to new CQ $\mathcal{Q}_2$:

$$q(p) \leftarrow \mathit{runs}(*,p),\ \mathit{in\_area}(p,a)$$

The final result of the rewriting procedure is $\mathcal{Q}_R = \mathcal{Q} \vee \mathcal{Q}_2$, i.e.

$$q(p) \leftarrow \mathit{in\_area}(p,a),\ \mathit{external}(e,a,p)$$
$$q(p) \leftarrow \mathit{runs}(*,p),\ \mathit{in\_area}(p,a)$$

We keep the original query since *external* may have initial (partial) data

Resolution looks for potential additional data

Why Stickiness?

- Stickiness for a set of TGDs guarantees BDDP

- Stickiness guarantees backward resolution-based query rewriting terminates

Applying resolution with a TGD without repeated marked variable, no new variable is introduced (but only new "don't care" symbols)

Example: Sticky set of TGDs:

$$\exists x, y, z \ emp(w, x, y, z) \leftarrow dept(v, w)$$

$$\exists z \ dept(w, z), runs(w, y), in\_area(y, x) \leftarrow emp(v, w, x, y)$$

$$\exists z \ external(z, y, x) \leftarrow runs(w, x), in\_area(x, y)$$

Query: $q(p) \leftarrow in\_area(p, a), external(e, a, p)$

Applying last TGD: $q(p) \leftarrow in\_area(p, a), runs(*, p), in\_area(p, a)$

Variables are all inherited from the first query Etc.

Example: Modification of previous one, for non-sticky case

$$\exists x, y, z \; emp(w, x, y, z) \leftarrow dept(v, w)$$
$$\exists z \; dept(w, z), runs(w, y), in\_area(y, x) \leftarrow emp(v, w, x, y)$$
$$\exists z, t \; external(z, y, t) \leftarrow runs(w, \underline{x}), in\_area(\underline{x}, y)$$

(x marked and occurs twice in body of last TGD)

Same query: $q(p) \leftarrow in\_area(p, a), external(e, a, p)$

Applying last TGD: $q(p) \leftarrow in\_area(p, a), runs(*, r), in\_area(r, a)$

Generates new, “relevant” variable r (in a join), not from the original query

(“Do not care variables” can get values in isolation)

Weakly-sticky programs relax conditions by ensuring “loose” join variables take only finitely many values, guaranteeing scenario (B) above

Conclusions

- Ontologies have been used for some time in AI (KR) and the Semantic Web

- Now they are being increasingly used in data management

  In particular, in interaction with relational DBs

- Ontologies can be used to access DBs through a model that is close to the user or application environment, e.g. business data

- They can also be used for data integration

- The ontological "schema" can be different from the DB schema

  Connection established via logical mappings

- DL and Datalog$\pm$ have been used for OBDA

- Datalog$\pm$ is a family of extensions of Datalog

  The latter has been around for more than two decades in the DB community

- DL and Datalog$\pm$ have been used to symbolically/logically represent ER, UML, ..., models

- Many applications are still to be unveiled

- There are many interesting open research problems

References

[1] M. Arenas, G. Gottlob, A. Pieris. Expressive Languages for Querying the Semantic Web. Proc. PODS 2014, pp. 14-26.

[2] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001, pp. 3443.

[3] A. Borgida and J. Mylopoulos: Data Semantics Revisited. Proc. SWDB, Springer LNCS 3372, 2004, pp. 9-26.

[4] A.Cali, G. Gottlob and Thomas Lukasiewicz: A General Datalog-Based Framework for Tractable Query Answering over Ontologies. *Journal of Web Semantics*, 2012, 14:57-83.

[5] A.Cali, G. Gottlob and A. Pieris. Towards More Expressive Ontology Languages: The Query Answering Problem. *Artificial Intelligence*, 2012, 193:87-128.

[6] A. Cali, G. Gottlob, G. Orsi and A. Pieris. Querying UML Class Diagrams. Proc. Foundations of Software Science and Computational Structures. Springer LNCS 7213, 2012, pp. 1-25.

[7] A.Cali, G. Gottlob and A. Pieris. Ontological Query Answering under Expressive Entity-Relationship Schemata. *Information Systems*, 2012, 37(4):320-335.

[8] A.Cali, G. Gottlob, Th. Lukasiewicz and A. Pieris. A Logical Toolbox for Ontological Reasoning. *SIGMOD Record*, 2011, 40(3):5-14.

[9] B. Chadrasekaran, J. Josephson and V.R. Benjamins. What are Ontologies, and Why Do We Need Them?. IEEE Intelligent Systems, Jan/Feb. 1999, pp. 20-26.

[10] G. Gottlob, G. Orsi and A. Pieris. Query Rewriting and Optimization for Ontological Databases. *ACM Trans. Database Syst.*, 2014, 39(3):25.

[11] D. Harel and B. Rumpe. Meaningful Modeling: What's the Semantics of "Semantics"?. *IEEE Computer*, 2004, 37(10): 64-72.

[12] P. Hitzler, M. Krötzsch and S. Rudolph. *Foundations of Semantic Web Technologies*. CRC Press, 2010.

[13] M. Milani, L. Bertossi and S. Ariyan. Extending Contexts with Ontologies for Multidimensional Data Quality Assessment. Proc. 5th International Workshop on Data Engineering meets the Semantic Web (DESWeb). Data Engineering Workshops (ICDEW), 2014, pp. 242 - 247.

[14] M. Lenzerini. Ontology-Based Data Management. Proc. AMW 2012, CEUR Proceedings, Vol. 866, pp. 12-15.

[15] A. Maedche, B. Motik, L. Stojanovic, R. Studer and R. Volz. Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems*, 2003, 18(2):26-33.

[16] N. Shadbolt, T. Berners-Lee and W. Hall. The Semantic Web Revisited. *IEEE Intelligent Systems*, 2006, 21(3):96-101.

[17] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *IEEE Computer*, 1992, 25(3):38-49.