Logic-based techniques for Information Integration

Marie-Christine ROUSSET
LIG
University of Grenoble

Information Integration

a difficult challenge!
Semantics: the glue between heterogeneous data sources

• Overview of some challenges and existing solutions for representing and exploiting the semantics
  – to describe and query heterogeneous pre-existing autonomous data sources

• Logic:
  – an appropriate formal background with associated automatic reasoning techniques

Focus on the use of logic for two primary challenges

(1) Describe and compare the content of pre-existing data sources

(2) Create single query interface over multiple and heterogeneous data sources
Illustration (Challenge1)

**Source 1:** Flights with atmost one Stop

**Source 2:** Direct Flights (without Stop)

**Source 3:** Flights whose Stop(s) are in AmericanCities only

**Source 4:** Flights with atleast one Stop in an AmericanCity

Only possible comparison resulting from the description in English:

Source 2 and Source 4 are disjoint

Other comparisons grounded on the logical semantics:

Source 1 $\cap$ Source 4 $\subseteq$ Source3 (under completeness assumption of Source 3)

Source 2 $\subseteq$ Source3 (under completeness assumption of Source 3 and depending on the logical interpretation of "whose …only")

Modeling in (description) logic

- Force to solve ambiguities and/or to set clear hypotheses by a set of formulas and axioms
- Gain automatic reasoning on those formulas and axioms

**Source 1:** The or Some Flights with atmost one Stop

Source1 $\equiv$ Flight $\wedge$ ($\leq$ 1 Stop)  versus Source1 $\not\subseteq$ Flight $\wedge$ ($\leq$ 1 Stop)

**Source 2:** Flights without Stop

Source2 $\subseteq$ Flight $\wedge$ ($\leq$ 0 Stop)

**Source 3:** The Flights whose Stops are in AmericanCities only

Source3 $\equiv$ Flight $\wedge$ $\forall$ Stop.AmericanCity

**Source 4:** Flights with atleast one Stop in an AmericanCity

Source4 $\subseteq$ Flight $\wedge$ $\exists$ Stop.AmericanCity
Woody Allen’s movies tonight in Paris, where, their reviews?

<table>
<thead>
<tr>
<th>Title</th>
<th>Actor</th>
<th>Director</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manhattan</td>
<td>Allen</td>
<td>Allen</td>
</tr>
<tr>
<td>Police</td>
<td>Depardieu</td>
<td>Godart</td>
</tr>
</tbody>
</table>

Internet Movie Data Base

Le Monde

The mediator approach

Mediated schema on cinema

Mediated schema on tourism

Query engine

Movie DB

Pariscpe

Le Monde

Air France

Relais&Chateaux

The mediator approach
Modeling and algorithmic issues

- Define a mediated schema
  - Structured vocabulary serving as a query interface for users queries
- Model the content of the sources to integrate in terms of the mediated schema
- Reformulate et decompose the users queries in queries executable against the data sources
- Combine the answers of local queries to build the answers of the global queries

Reformulation of the query: illustration
The underlying machinery is logic-based

Plan

- Logical foundations of databases
  - for modeling schemas, constraints, queries, views
  - for query containment and rewriting

- Description logics in a nutshell for reasoning on data semantics

- Models and algorithms for (virtual) integration of heterogeneous data sources
Logic: a unifying framework for posing and solving database problems

Queries

- Open formula of first-order logic (FOL)

\[ q(x): \exists y \, \Phi(x,y) \]

- \( x \) is a vector of free (distinguished) variables
- \( \Phi(x,y) \) is a formula the free variables of which are those of \( x \) and \( y \) (with possibly constants and bound variables)

- Example (conjunctive query)

\[ q(X) : \exists A,C \, \text{Flight}(X) \land \text{ArrivalAirport}(X,A) \land \text{Located}(A,C) \land \text{Capital}(C) \]

Datalog notation:

\[ q(X) \leftarrow \text{Flight}(X) \land \text{ArrivalAirport}(X,A) \land \text{Located}(A,C) \land \text{Capital}(C) \]
Logical semantics

• Answers to a query \( q \) relatively to a KB \( K \)
  \[ \text{Ans}(q,K) = \{ a \mid K \models q(a) \} \]
  – \( K \) is a set of closed formulas
    • A DB extension (a set of facts \( R(a_1, \ldots, a_n) \) where \( R \) is a relation of the schema) + possibly constraints
    • Abox ∪ Tbox (a KB expressed in Description Logic)
  – \( a \) is a vector of constants appearing in \( K \)
  – \( q(a) \): obtained from \( \exists y \Phi(x,y) \) by replacing variables of \( x \) with constants of \( a \)
  \( K \models q(a) \): the tuple \( a \) satisfies the query \( q \) in all the interpretations satisfying (models of) \( K \)

Interpretations of formulas in logic

• an interpretation \( I \) of \( \varphi \):
  – A domain of interpretation \( \Delta^I \)
  – A function of interpretation mapping
    • constants \( a \) in \( \varphi \) to elements \( a^I \) of \( \Delta^I \)
    • (functions \( f \) and relations \( R \)) in \( \varphi \) to (functions \( f^I \) from \( \Delta^I \) to \( \Delta^I \)) to relations \( R^I \) on \( \Delta^I \)
  – Rules of interpretation for interpreting any formula from the interpretation of its sub-formulas
    • Interpretation of a closed formula: true or false
    • Interpretation of an open formula with \( n \) free variables: an \( n \)-ary relation on \( \Delta^I \)
• A model of a formula \( \varphi \): an interpretation \( I \) such that
  – \( \varphi^I = \text{true} \) (if \( \varphi \) is closed)
  – \( \varphi^I \neq \emptyset \) (if \( \varphi \) is open)
**Rules of interpretation for quantifiers**

- Let $\varphi$ be a closed formula of the form $\forall x \psi$
  
  $[\forall x \psi(x)]^I = \text{true iff for every } e \in \Delta^I, \psi^I(e) \text{ is true}$

- Let $\varphi$ be a closed formula of the form $\exists x \psi$
  
  $[\exists x \psi(x)]^I = \text{true iff there exists } e \in \Delta^I, \psi^I(e) \text{ is true}$

- Let $\varphi(x_1,\ldots,x_n)$ be a formula with $n$ free variables
  
  $[\varphi(x_1,\ldots,x_n)]^I = \{(e_1,\ldots,e_n) \in \Delta^I \times \ldots \times \Delta^I / \varphi^I(e_1,\ldots,e_n) \text{ is true}\}$

---

**$K |= q(a)$: particular case**

**$K$ is a DB extension, $q$ a positive formula**

- **$K$ can be viewed as an Herbrand model**
  - $\Delta^I = \text{the set of all the constants in the DB extension}$
  - $a^I = a$ for every constant $a$
  - $R^I = R$ for every relation $R$

- **$K |= q(a)$ is reduced to evaluate $q(a)$ in $K$**

- **If $q$ is a conjunctive query**
  - $K |= q(a)$ is true iff there exists a mapping $m$ from the constants and existential variables of $q(a)$ to constants in $K$ such that for every conjunct $R(t_1,\ldots,t_n)$ of $q(a)$: $R(m(t_1),\ldots,m(t_n)) \text{ is in } K$

Homomorphism theorem in [Chandra-Merlin 77]:
*Optimal implementation of conjunctive queries in relational database*
9th ACM symposium on Theory of Computing (STOC’77)
Reasoning problems in FOL

• Satisfiability checking of a formula or a set of formulas
  – existence of a model
• Logical entailment: $\varphi_1, \ldots, \varphi_n \models \varphi$
  – Every model of $\varphi_1, \ldots, \varphi_n$ is a model of $\varphi$
  – Can be reduced to satisfiability checking:
    $\varphi_1, \ldots, \varphi_n, \neg \varphi$ is unsatisfiable
• Semi-decidable problems
  – There does not exist an algorithm to decide whether any
    formula is satisfiable or not (is entailed or not by a given set of
    formulas)
  – Infinite number of interpretations of a FOL formula

Query evaluation

• A reasoning problem
  \[ q(x) : \exists y \Phi(x,y) \]
  \[ \text{Ans}(q,K) = \{ a | K \models q(a) \} \]
• Decidable case
  \[ K = \text{DB extension} \]
  \[ q : \text{conjunctive query} \]
  \[ q(X) : \exists A,C \text{ Flight}(X) \land \text{ArrivalAirport}(X,A) \land \text{Located}(A,C) \land \text{Capital}(C) \]
  – polynomial in the size of the data, NP-complete in the size of the
    query
    (results from the homomorphism theorem [Chandra-Merlin 77])
  – Optimized algorithms for efficient computation of \( \text{Ans}(q,K) \) in
    DBMSs
Query containment

• Let \( q_1(x) : \exists y_1 \Phi_1(x,y_1) \) and \( q_2(x) : \exists y_2 \Phi_2(x,y_2) \)

\[ q_1 \subseteq q_2 \iff \text{Ans}(q_1, I(DB)) \subseteq \text{Ans}(q_2, I(DB)) \text{ for every } I(DB) \]

• Another reasoning problem:

\[ \exists y_1 \Phi_1(x,y) \models \exists y_2 \Phi_2(x,y) ? \]

Particular case: containment of conjunctive queries

• NP-complete problem
• Algorithm illustrated on an example:
  \( q_1(X): R(X,Y), R(Y,Z), R(Z,Z) \)
  \( q_2(X'): R(X',Y'), R(Y',Z'1), R(Y,Z'2) \)
  – \( q_1 \) viewed as a DB extension by freezing its variables: \( X, Y \) and \( Z \) considered as constants
  – evaluating \( q_2 \) against this DB « extension »
    • If \( X \) is an answer: YES
    • If not: NO (\( q_1 \) is not contained in \( q_2 \))
    – On the example: \( X \) is an answer and thus \( q_1 \subseteq q_2 \)
No containment: example

\[ q_1(X): R(X,Y), R(Y,Z), R(Z,Z) \]
\[ q_2(X'): R(X',Y'), R(Y',Z'1), R(Y',Z'2) \]
- \( q_2 \not\subset q_1 \)
- Freezing the variables of \( q_2 \):
  \( X', Y', Z'1, Z'2 \) are distinct constants
- Evaluation of \( q_1 \) against this \( D \) « extension »
  - \( \text{Ans}(q_1, \text{freeze}(q_2)) = \emptyset \) thus: \( q_2 \not\subset q_1 \)

Query containment

- Central problem for the comparison of different data sources
  - A query: a formula that describes in a compact way the content of a data source
- Other decidable cases:
  - When the queries are expressible in Description Logic
  - Query containment = subsomption between two concept descriptions
  - Extended with constraints on the schema: inclusion statements between concept expressions
Description logics in a nutshell for reasoning on data semantics

Description Logics

• Logic-based representation of classes of objects using a set of constructors (having a FOL semantics)
  – Decidable fragments of FOL based on unary and binary predicates
    • Unary predicates: classes (called concepts)
    • Binary predicates: properties (called roles)
• Many decidability and complexity results for reasoning problems
• Implemented reasoners: RACER, PELLET
Description Logics by example

the description :
\( \text{Paper} \cap (\exists \text{ Author } \text{PhDStudent}) \cap (\exists \text{ Author } (\neg \text{PhDStudent})) \)

is subsumed by : \( \text{Paper} \cap (\geq 2 \text{ Author}) \)

is disjoint with: \( \text{Paper} \cap (\forall \text{ Author } \text{PhDStudent}) \)

Description Logics by example

the query / the source content :
\( \text{Paper} \cap (\exists \text{ Author } \text{PhDStudent}) \cap (\exists \text{ Author } (\neg \text{PhDStudent})) \)
\{ \text{x} | \text{Paper}(\text{x}) \land \exists \exists \exists \exists \text{y} (\text{Author}(\text{x},\text{y}) \land \text{PhDStudent } (\text{y})) \land \exists \exists (\text{Author} \ (\text{x},\text{z}) \land \neg \text{PhDStudent}(\text{z})) \}

is contained in : \( \text{Paper} \cap (\text{atleast 2 Author}) \)
\{ \text{x} | \text{Paper}(\text{x}) \land \exists \exists \exists \exists \text{y} (\text{Author}(\text{x},\text{y}) \land (\text{Author} \ (\text{x},\text{z}) \land (y \neq z))) \}

is disjoint with: \( \text{Paper} \cap (\forall \text{ Author } \text{PhDStudent}) \)
\{ \text{x} | \text{Paper}(\text{x}) \land (\forall \text{y} \ (\text{Author}(\text{x},\text{y}) \Rightarrow \text{PhDStudent}(\text{y})) \} \)

(restricted) negation
FOL semantics of the main constructors

\[(C_1 \cap C_2)(X) \equiv C_1(X) \land C_2(X)\]
\[(C_1 \cup C_2)(X) \equiv C_1(X) \lor C_2(X)\]
\[(\forall R \, C)(X) \equiv \forall Y \, (R(X,Y) \Rightarrow C(Y))\]
\[(\exists R \, C)(X) \equiv \exists Y \, (R(X,Y) \land C(Y))\]
\[(\geq n \, R)(X) \equiv \exists Y_1 \ldots Y_n \, (R(X, Y_1) \land \ldots \land R(X, Y_n) \land \land_{\{i,j \mid i \neq j\}} Y_i \neq Y_j)\]
\[(\leq n \, R)(X) \equiv \forall Y_1 \ldots Y_{n+1} \, (R(X, Y_1) \land \ldots \land R(X, Y_{n+1}) \land \Rightarrow \land_{\{i,j \mid i \neq j\}} Y_i = Y_j)\]

A Description Logic KB

**Abox A : a set of facts**
Professor(Jim) \ HasTutor(John, Mary) \ TeachesTo(John, Bill)

**Tbox T : a set of General Concept Inclusions (CGI)**
Professor \subseteq \exists TeachesTo
Student \subseteq \exists HasTutor
\exists TeachesTo \subseteq Student
\exists HasTutor \subseteq Professor
Professor \subseteq \neg Student
Reasoning problems

• Subsumption checking
  – Between two concept descriptions
  – Between two concept descriptions given a set of GCIs
    defined in a Tbox T:
    \[ T \models C_1 \subseteq C_2 ? \]

• Membership checking of an instance to a concept
  – Given a concept C, a constant a, a Tbox T, an Abox A (a set
    of facts of the form A(b) and P(b,c))
    \[ T \cup A \models C(a) ? \]

• Many decidability and complexity results in
  function of the constructors and the GCIs allowed
  in T

Some results of complexity

<table>
<thead>
<tr>
<th>Constructors</th>
<th>Complexity of subsumption checking</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALN ((\cap, \forall, \geq, \leq))</td>
<td>P</td>
</tr>
<tr>
<td>ALE ((\cap, \forall, \exists))</td>
<td>NP-complet</td>
</tr>
<tr>
<td>ALNE ((\cap, \forall, \exists, \geq, \leq))</td>
<td>NP-complet</td>
</tr>
<tr>
<td>ALN + conjunction of roles</td>
<td>co-NP-hard</td>
</tr>
<tr>
<td>ALC ((\cap, \forall, \neg))</td>
<td>Pspace-complet</td>
</tr>
<tr>
<td>DL-Lite (GCIs with restriction on (\neg, \exists))</td>
<td>P</td>
</tr>
</tbody>
</table>
Example of constraints expressible in DL-Lite

Professor ⊆ ∃TeachesTo PI
Student ⊆ ∃HasTutor PI
∃TeachesTo ⊆ Student PI
∃HasTutor ⊆ Professor PI
Professor ⊆ ¬Student NI

HasTutor ⊆ TeachesTo PI

Expressivity of DL-Lite

• Captures the main constraints used in DB and Software Engineering
  – Relation ISA : A1 ⊆ A2
  – Disjunction : A1 ⊆ ¬A2
  – typing :
    • ∃P ⊆ A1 (the first attribute of P is typed by A1)
    • ∃P ⊆ A2 (the second attribute of P is typed by A2)
  – Mandatory or forbidden properties :
    • A ⊆ ∃P A ⊆ ∃P A ⊆ ¬∃P A ⊆ ¬∃P

• Extends RDFS and corresponds to a profile of OWL2
  – Languages for describing metadata and ontologies
  – W3C standards for the Semantic Web
DL used for reasoning on data

- Comparing different data sources described using DL
  - Inclusion, disjointness
- Checking query containment in presence of constraints on the schema
- Checking data consistency
- Checking that there exists an answer for a query without evaluating it
- Reformulating queries (generalization, specialization)
- DL reasoning is « open-world »: the data are incomplete
  - It is the case for Web data (in contrast with DBMS)

Close versus Open World

- Close World
  - Constraints are used to check consistency but also the completeness of the DB

  Professor $\subseteq \exists$TeachesTo

  Referential constraint: all the constants of the table Professor must appear in the table TeachesTo

  - Constraints are then not used to compute the answers
- Open World
  - Constraints are additional knowledge on the (incomplete) data declared in the DB
  - They can be used in the reasoning underlying the query evaluation
Example

K:

<table>
<thead>
<tr>
<th>Professor</th>
<th>TeachesTo</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mary</td>
<td>Bill</td>
</tr>
<tr>
<td>Jim</td>
<td>John</td>
</tr>
<tr>
<td>Tom</td>
<td>Ann</td>
</tr>
</tbody>
</table>

+ Professor \( \subseteq \exists \text{TeachesTo} \)

\[ q(x): \text{TeachesTo}(x,y) \]

\[ \text{Ans}(q,K) = \{ \text{Mary, John, Jim, Tom} \} \]

Answering queries in Open World

- Problem in general as hard as reasoning in FOL or in fragments of FOL
  - For each tuple \( \mathbf{a} \) of constants, \( K \models q(\mathbf{a}) \) ?
- Decidable fragments for which \( \text{Ans}(q,K) \) is polynomial in the size of the data
  - Deductive DB: \( K = \text{DB} + \text{rules} \)
  - DL-Lite: \( K = \text{Abox} + \text{Tbox} \)
- Common approach:
  - reformulation of the query in a union of queries that are directly executable against a DB
Illustration in DL-Lite

\textbf{Abox A :}
Professor(Jim)\quad HasTutor(John, Mary)\quad TeachesTo(John, Bill)

\textbf{Tbox T :}
Professor \subseteq \exists \text{TeachesTo}
Student \subseteq \exists \text{HasTutor}
\exists \text{TeachesTo} \cdot \subseteq \text{Student}
\exists \text{HasTutor} \cdot \subseteq \text{Professor}
Professor \subseteq \neg \text{Student}

\textbf{Conjunctive queries on concepts and atomic roles :}
\begin{align*}
q_0(x) & \leftarrow \text{TeachesTo}(x,y)\wedge\text{HasTutor}(y,z)
\end{align*}

Query reformulation

- Reformulation algorithm: illustration
  \begin{align*}
  q_1(x) & \leftarrow \text{TeachesTo}(x,y)\wedge\text{Student}(y) \\
  q_2(x) & \leftarrow \text{TeachesTo}(x,y)\wedge\text{TeachesTo}(z',y) \\
  q_3(x) & \leftarrow \text{TeachesTo}(x,y') \\
  q_4(x) & \leftarrow \text{Professor}(x) \\
  q_5(x) & \leftarrow \text{HasTutor}(u,x)
  \end{align*}
- For each i: \( q_i, T \models q_0 \)
- \( \text{Ans}(q_0, T \cup A) = \bigcup_i \text{Ans}(q_i, A) \)
- Sound algorithm if \( T \cup A \) is satisfiable
Illustration

Abox A :
Professor(Jim)  HasTutor(John, Mary)  TeachesTo(John, Bill)

Query:  q0(x) ← TeachesTo(x,y)\land HasTutor(y,z)

Reformulations:
q1(x) ← TeachesTo(x,y)\land Student(y)
q2(x) ← TeachesTo(x,y)\land TeachesTo(z',y)
q3(x) ← TeachesTo(x,y')
q4(x) ← Professor(x)
q5(x) ← HasTutor(u,x)

Ans(q,A\cup T) = \{Mary, Jim, John\}

Satisfiability checking

• T \cup A may be unsatisfiable

Abox A :
Professor(Jim)  HasTutor(John, Mary)  TeachesTo(John, Bill)

Tbox T :
Professor \subseteq \exists TeachesTo
Student \subseteq \exists HasTutor
\exists TeachesTo' \subseteq Student
\exists HasTutor' \subseteq Professor
Professor \subseteq \neg Student  +  \exists TeachesTo \subseteq \neg Student
\exists HasTutor \subseteq Student
Satisfiability checking in DL-Lite

- Saturation of the Negative Inclusions (NIs) that are translated in boolean conjunctive queries evaluated against the Abox seen as a DBMS
  - True iff \( A \cup T \) unsatisfiable

**Abox A:**
Professor(Jim) HasTutor(John, Mary) TeachesTo(John, Bill)

**Tbox T:**
\( \exists \text{TeachesTo} \subseteq \neg \text{Student} \)
\( \exists \text{HasTutor} \subseteq \text{Student} \)
\( \exists \text{TeachesTo} \subseteq \neg \exists \text{HasTutor} \)

**Q:** TeachesTo(X,Y) \( \land \) \( \exists \text{HasTutor}(X,Y') \)

References

Models and algorithms for (virtual) integration of heterogeneous data sources

The mediator approach
Underlying principles

• Defining a mediated schema (also called a global schema) : serving as query interface for users

• Specifying schema mappings between the global schema and the schemas of the local data sources
  – Global-As-Views (GAV) approach: the global relations are defined as views over the local relations
  – Local-As-Views (LAV) approach: the local relations are defined as views over the global relations

• Rewriting the users queries (expressed using global relations) in terms of local relations => logical query plan

Views

• Named queries that can be re-used in other queries
  – Represents by a formula the answer set of a query or the content of a data source

• Example
  
  Source1(X,Y1,Y2) : Flight(X) ∧ DepartureAirport(X,Y1) ∧ ArrivalAirport(X,Y2)
  Source2(X,Y) : Place(X) ∧ Located(X,Y) ∧ Capital(Y)

• Can be materialized or virtual
  – Their extension is stored (in memory or in a cache) or computed on demand (by querying a data source)
Different semantics

of the correspondence $v(x) : \text{def}(x,y)$ between the view and the query defining it:

- « exact » semantics
  \[ \text{Ext}(v) = \text{Ans}(\text{def},K) \]
  \text{axiom : } \forall x \ [v(x) \Leftrightarrow \exists y \ \text{def}(x,y)] \text{ added to } K

- « sound » semantics
  \[ \text{Ext}(v) \subseteq \text{Ans}(\text{def},K) \]
  \text{axiom : } \forall x \ [v(x) \Rightarrow \exists y \ \text{def}(x,y)] \text{ added to } K

- « complete » semantics
  \[ \text{Ans}(\text{def},K) \subseteq \text{Ext}(v) \]
  \text{axiom : } \forall x \ [\exists y \ \text{def}(x,y) \Rightarrow v(x)] \text{ added to } K

The Global-As-Views approach
Illustration on 4 existing data sources

- S1: a catalogue of teaching programs of (some) French universities
  \[ S1.Catalogue(nomUniv, programme) \]
- S2: Erasmus students enrolled in courses of (some) European universities
  \[ S2.Erasmus(student, course, univ) \]
- S3: Foreign students enrolled in programs of (some) French universities
  \[ S3.CampusFrance(student, program, university) \]
- S4: the course content of (some) international master programs
  \[ S4.Mundus(programTitle,course) \]

GAV modeling of a mediated schema

**University (U):** \[ S1.Catalogue(U,P) \lor S2.Erasmus(N,C,U) \lor S3.CampusFrance(N',P',U) \]

**MasterStudent (N):** \[ S2.Erasmus(N,C,U), S4.Mundus(P,C) \lor S3.CampusFrance(N',P',U'),S4.Mundus(P',C') \]

**MasterCourse (C):** \[ S4.Mundus(P,C) \]

**MasterProgram(P):** \[ S4.Mundus(P,C) \]

**EnrolledIn (N,P):** \[ S2.Erasmus(N,C,U), S4.Mundus(P,C) \lor S3.CampusFrance(N,P,U'),S4.Mundus(P,C') \]

**RegisteredTo(N,U):** \[ S3.CampusFrance(N,P,U) \]
Logical semantics of GAV mappings

\[
\text{MasterStudent (N)} : \ S2.\text{Erasmus}(N,C,U), \ S4.\text{Mundus}(P,C) \\
\lor \ S3.\text{CampusFrance}(N,P',U'), S4.\text{Mundus}(P',C')
\]

Exact semantics:

\[
\forall N \left[ \exists C \exists U \exists P \left( S2.\text{Erasmus}(N,C,U) \land S4.\text{Mundus}(P,C) \right) \\
\lor \left( \exists C' \exists U' \exists P' \left( S3.\text{CampusFrance}(N,P',U'), S4.\text{Mundus}(P',C') \right) \right) \right] \\
\iff \text{MasterStudent (N)}
\]

Sound semantics:

\[
\forall N \left[ \exists C \exists U \exists P \left( S2.\text{Erasmus}(N,C,U) \land S4.\text{Mundus}(P,C) \right) \\
\lor \left( \exists C' \exists U' \exists P' \left( S3.\text{CampusFrance}(N,P',U'), S4.\text{Mundus}(P',C') \right) \right) \right] \\
\Rightarrow \text{MasterStudent (N)}
\]

Query rewriting by unfolding

The two semantics express how to obtain tuples for the corresponding global relation

\[
\Rightarrow \text{The logical query plans are obtained by } \text{unfolding} \text{ each atom of the query, i.e., by replacing each atom that can be matched with the head of atleast one view with the body of the corresponding view (possiblly splitted in conjunctive views)}
\]

\[
\text{MasterStudent (N)} : \ S2.\text{Erasmus}(N,C,U), \ S4.\text{Mundus}(P,C) \\
\text{MasterStudent (N)} : \ S3.\text{CampusFrance}(N,P',U'), S4.\text{Mundus}(P',C')
\]
Illustration

Query: \( q(x): \) RegisteredTo\((s,x)\), MasterStudent\((s)\)

Conjunctive views:

- RegisteredTo\((N,U)\): \( S3.\text{CampusFrance}(N,P,U) \)
- MasterStudent \((N)\) : \( S2.\text{Erasmus}(N,C,U), S4.\text{Mundus}(P,C) \)
- MasterStudent \((N)\) : \( S3.\text{CampusFrance}(N,P',U'), S4.\text{Mundus}(P',C') \)

2 rewritings by unfolding:

(existential variables in the view bodies are replaced by new variables)

u1\((x)\):
\[
S3.\text{CampusFrance}(s,v1,x), S2.\text{Erasmus}(s,v2,v3), S4.\text{Mundus}(v4,v2)
\]

u2\((x)\):
\[
S3.\text{CampusFrance}(s,v5,x), S3.\text{CampusFrance}(s,v6,v7), S4.\text{Mundus}(v6,v8)
\]

Illustration (ctd)

Simplification of \( u2(x) \):

\[
S3.\text{CampusFrance}(s,v5,x), S3.\text{CampusFrance}(s,v6,v7), S4.\text{Mundus}(v6,v8)
\]

by unifying the two first atoms into \( S3.\text{CampusFrance}(s,v6,x) \)

with the substitution \( \sigma = \{ v5/v6, v7/x \} \) where \( v5 \) and \( v7 \) are

unbounded existential variables

\( \Rightarrow \) equivalent query expression

2 resulting logical query plans:

- \( u1(x)\):
  \[
  S3.\text{CampusFrance}(s,v1,x), S2.\text{Erasmus}(s,v2,v3), S4.\text{Mundus}(v4,v2)
  \]

- \( u'2(x)\): \( S3.\text{CampusFrance}(s,v6,x), S4.\text{Mundus}(v6,v8) \)
Results and discussion

• The union $U$ of the logical query plans obtained by unfolding the atoms of a query $q$ using a set $GV$ of GAV mappings is complete: for every instance $I$ of the source relations, $\text{ans}(q, GV \cup I) = \bigcup_{u \in U} \text{ans}(u, I)$

• The evaluation of some query plans may lead to redundant answers or to no answer at all
  – It can be known in advance (before their execution) if some additional knowledge is provided
  – Example: from the knowledge that the students found in S3. CampusFrance are non European Students, while those found in S2.Erasmus are European students, we can infer that the query plan $u_1$ will return an empty set of answers

  $u_1(x): S3.CampusFrance(s,v1,x), S2.Erasmus(s,v2,v3), S4.Mundus(v4,v2)$

Main limitation of the GAV approach

• Adding or removing data sources requires to revise all the GAV mappings defining the global schema
  – when a new data source arrives, we must consider how it may be combined with all the existing data sources to produce tuples of any global relation

$\Rightarrow$ In the Local-As-Views (LAV) approach, the mediated schema is designed to remain stable even when data sources join or leave the integration system
The LAV approach

• The mediated schema is defined as a set of global relations in function of a given domain

• Example:
  
  \[
  \begin{align*}
  \text{Student(studentName)} & , \ldots, \text{University(uniName)} \\
  \text{Program(title)}, \text{MasterProgram(title)}, \text{Course(code)} \\
  \text{EnrolledInProgram(studentName,title)} \\
  \text{EnrolledInCourse(studentName,code)}, \text{PartOf(code,title)} \\
  \text{RegisteredTo(studentName, uniName)} \\
  \text{OfferedBy(title, uniName)}
  \end{align*}
  \]

LAV mappings

S1. Catalogue(U,P):
  
  \[
  \begin{align*}
  \text{FrenchUniversity(U)}, \text{Program(P)}, \\
  \text{OfferedBy(P,U)}, \text{OfferedBy(P',U)}, \text{MasterProgram(P')}
  \end{align*}
  \]

S2. Erasmus(S,C,U):
  
  \[
  \begin{align*}
  \text{Student(S)}, \text{EnrolledInCourse(S,C)}, \text{PartOf(C,P)}, \\
  \text{OfferedBy(P,U)}, \text{EuropeanUniversity(U)}, \text{RegisteredTo(S,U')} \\
  \text{EuropeanUniversity(U')}, \text{U≠U'}
  \end{align*}
  \]

S3. CampusFrance(S,P,U):
  
  \[
  \begin{align*}
  \text{NonEuropeanStudent(S)}, \text{EnrolledInProgram(S,P)}, \\
  \text{Program(P)}, \text{OfferedBy(P,U)}, \text{FrenchUniversity(U)}, \\
  \text{RegisteredTo(S,U)}
  \end{align*}
  \]

S4. Mundus(P,C):
  
  \[
  \begin{align*}
  \text{MasterProgram(P)}, \text{OfferedBy(P,U)}, \text{OfferedBy(P,U')}, \\
  \text{EuropeanUniversity(U)}, \text{NonEuropeanUniversity(U)}, \text{PartOf(C,P)}
  \end{align*}
  \]
Logical semantics of the LAV mappings

S1.Catalogue(U,P):
FrenchUniversity(U), Program(P),
OfferedBy(P,U), OffereBy(P',U), MasterProgram(P')

Exact semantics:
∀U ∀P [S1.Catalogue(U,P)
⇒ ∃P' (FrenchUniversity(U), Program(P),
OfferedBy(P,U), OffereBy(P',U), MasterProgram(P'))]

Sound semantics:
∀U ∀P [S1.Catalogue(U,P)
⇔ ∃P' (FrenchUniversity(U), Program(P),
OfferedBy(P,U), OffereBy(P',U), MasterProgram(P'))]

Discussion

• Allows a fine-grained description of the data sources, and a loose coupling between local and global relations
  – Important for robustness and flexibility
  • Illustration: if we are interested in Master students, we do not need to know in advance how to join the available data sources to obtain them like in the GAV approach; we just define them as a global query
    MasterStudent(S):
    Student(S), EnrolledInProgram(S,P), MasterProgram(P)

• Price to pay flexibility and robustness: building the rewritings requires more work than the simple unfolding of the GAV approach
  – Several algorithms: Bucket, Minicon, Inverse-rules
The Bucket algorithm

• Input
  – A conjunctive query (with comparison predicates) over a global schema
  – A set of local relations defined as conjunctive views (with comparison predicates) « sound» semantics) over the global schema
• output : a set of conjunctive queries over the local relations
• Implemented in Information Manifold


Principle: two steps

• Create a « bucket» for each atom g of the query
  – Store each view atom with an atom in its definition being unifiable with g (without violating comparison predicates)
• Build the set of candidate rewritings
  – Take one view-atom in each bucket and take their conjunction
  – For each candidate rewriting, check if its expansion is contained in the query
    • If yes: return it in the output
    • If no : try to add some comparison predicates to satisfy the containment
Creation of the buckets: illustration

q(x): RegisteredTo(s,x), EnrolledInProgram(s,p), MasterProgram(p)

- \textit{RegisteredTo}(S, U') is in the definition of S2.Erasmus(S,C,U)
  but mapping the existential variable U' in the view definition to the
distinguished variable x in the query is not enough to infer
\textit{RegisteredTo}(s,x) from S2.Erasmus(s,C,U)
S2.Erasmus(s,C,U) is not added to \text{Bucket}(\text{RegisteredTo}(s,x))

- \textit{RegisteredTo}(S, U) in the definition of S3.CampuFrance(S,P,U)
  has U as distinguished variable to which the distinguished
variable x can be mapped
\text{Bucket}(\text{RegisteredTo}(s,x)) = \{S3.CampusFrance(s, v1,x)\}

Combination of the buckets

\begin{align*}
\text{Bucket}(\text{RegisteredTo}(s,x)) &= \{S3.CampusFrance(s, v1,x)\} \\
\text{Bucket}(\text{EnrolledInProgram}(s,p)) &= \{S3.CampusFrance(s, p,v2)\} \\
\text{Bucket}(\text{MasterProgram}(p)) &= \{S1.Catalogue(v3,v4), S4.Mundus(p,v5)\}
\end{align*}

⇒ 2 \textbf{candidate} rewritings :

r1(x): S3.CampusFrance(s, v1,x), S3.CampusFrance(s, p,v2),
S1.Catalogue(v3,v4)
r2(x): S3.CampusFrance(s, v1,x), S3.CampusFrance(s, p,v2),
S4.Mundus(p,v5)
Complexity

The creation of buckets:
\[ O(N \times M \times V) \]
\( N = \) size of the query, \( V = \) number of views, \( M = \) size of the views

\( \Rightarrow \) \( N \) buckets containing each \( O(M \times V) \) view atoms

\( \Rightarrow \) The number of candidate rewritings: \( O((M \times V)^N) \)

Verification of each candidate rewriting

\( q(x): \) \text{RegisteredTo}(s,x), \text{EnrolledInProgram}(s,p), \text{MasterProgram}(p) \\
\( r_1(x): \) \text{S3.CampusFrance}(s,v_1,x), \text{S3.CampusFrance}(s,p,v_2), \text{S1.Catalogue}(v_3,v_4) \\

\( r_1(x) \) is a \textbf{valid} rewriting

iff \( r_1(x) \) together with the LAV mappings logically entail \( q(x) \)

iff the \textbf{expansion} of \( (r_1(x)) \) is \textbf{contained} in \( q(x) \)
Verification by expansion and containment checking

\[ q(x): \text{RegisteredTo}(s,x), \text{EnrolledInProgram}(s,p), \text{MasterProgram}(p) \]

\[ r1(x): \text{S3.CampusFrance}(s, v1,x), \text{S3.CampusFrance}(s, p,v2), \text{S1.Catalogue}(v3,v4) \]

Expand\((r1(x))\): NonEuropeanStudent\((s)\), EnrolledInProgram\((s,v1)\), Program\((v1)\), OfferedBy\((v1,x)\), FrenchUniversity\((x)\), RegisteredTo\((s,x)\), EnrolledInProgram\((s,p)\), Program\((p)\), OfferedBy\((p,v2)\), FrenchUniversity\((v2)\), RegisteredTo\((s,v2)\), FrenchUniversity\((v3)\), Program\((v4)\), OfferedBy\((v4,v3)\), OfferedBy\((v5,v3)\), MasterProgram\((v5)\)

Expand\((r1(x))\) is not contained in \(q(x)\): \(r1\) is not a valid rewriting

Minicon: optimization of Bucket

- Containment checking is avoided by a stricter verification of the atoms to add to the buckets
  - When the definition of a view \(V\) contains an atom \(g'\) such that: \(\sigma(g') = g\)
    - If an existential variable \(Y\) of \(g\) appears in other atoms \(g_1, g_2, \ldots, g_k\) of the query
    - If \(Y' = \sigma(Y)\) is also existential in the view definition
      - \(\sigma(V)\) is added to Bucket\((g)\) only if \(g_1, g_2, \ldots, g_k\) are also covered by the definition of \(\sigma(V)\)
Illustration

\( V_4(X) : \text{cite}(X,Y), \text{cite}(Y,X) \)

\( V_5(X,Y) : \text{sameTopic}(X,Y) \)

\( V_6(X,Y) : \text{cite}(X,Z), \text{cite}(Z,Y), \text{sameTopic}(X,Z) \)

Query : \( Q(U) : \text{cite}(U,V), \text{cite}(V,U), \text{sameTopic}(U,V) \)

Bucket (\text{cite}(U,V)) ?

- \( V_4(U) \) is not added because sameTopic(U,V) is not covered by the definition of \( V_4(U) \)
- \( V_6 ? \)

\( \sigma(X)=U \) et \( \sigma(Z)=V \)

Covering of \( \text{cite}(V,U) \) by the definition of \( V_6(U,Y) \) => \( \sigma(Y)=U \)

Covering of \( \text{sameTopic}(U,V) \) by the definition of \( \sigma(V_6(X,Y)) \) ? yes

\[ \text{Bucket(cite}(U,V)) = \{V_6(U,U)\} \]

\[ \text{cover}(V_6(U,U)) = \{\text{cite}(U,V), \text{cite}(V,U), \text{sameTopic}(U,V)\} \]

\[ \Rightarrow r(U) : \text{V}_6(U,U) \text{ is a valid rewriting of } Q(U) \]

Advantages of Minicon

- The rewritings are directly obtained by taking the conjunction of the view-atoms in the « buckets » which have pairwise disjoint coverings

- Results

  – theoretical :
    - same worst-case complexity as Bucket (exponential in the size of the query)

  – experimental :
    - Scalable when there are many views
The Inverse-rules algorithm

- Principle:
  - The LAV mappings are split into GAV mappings (called inverse rules)
    
    independently of the query
    
    - Existential variables are replaced by Skolem terms in order to keep the binding of the different occurrences of existential variables
  - At query time, the rewritings are obtained by unfolding
    
    - The unfolding operation is a little trickier because of the Skolem functions

Illustration

\[ V4(X) : \text{cite}(X,Y), \text{cite}(Y,X) \]
\[ V5(X,Y) : \text{sameTopic}(X,Y) \]
\[ V6(X,Y) : \text{cite}(X,Z), \text{cite}(Z,Y), \text{sameTopic}(X,Z) \]

Result of the Inverse-rules algorithm:

\[ \text{cite}(X,f1(X)) : V4(X) \]
\[ \text{cite}(f1(X),X) : V4(X) \]
\[ \text{sameTopic}(X,Y) : V5(X,Y) \]
\[ \text{cite}(X,f2(X,Y)) : V6(X,Y) \]
\[ \text{cite}(f2(X,Y),X) : V6(X,Y) \]
\[ \text{sameTopic}(X,f2(X,Y)) : V6(X,Y) \]
Query unfolding (illustration)

Q(U):
\(\text{cite}(U,V), \text{cite}(V,U), \text{sameTopic}(U,V)\)
\(\sigma = \{X/U, V/f1(U)\}\)
Q'1(U):
\(\text{V4}(U), \text{cite}(f1(U), U), \text{sameTopic}(U, f1(U))\)
Q'2(U):
\(\text{V4}(U), \text{V4}(U), \text{sameTopic}(U, f1(U))\)
Q'3(U):
\(\text{V4}(U), \text{V5}(U,f1(U))\)

The evaluation of this query plan will produce no answer: there is no way to match \(\text{V5}(U,f1(U))\) with a fact \(\text{V5}(a,b)\) in the data source.

Query unfolding (illustration ctd)

Q(U):
\(\text{cite}(U,V), \text{cite}(V,U), \text{sameTopic}(U,V)\)
\(\sigma = \{X/U, V/f2(U,Y)\}\)
Q''1(U):
\(\text{V6}(U,Y), \text{cite}(f2(U,Y), U), \text{sameTopic}(U, f2(U,Y))\)
\(\sigma = \{X/U, Y/U\}\)
Q''2(U):
\(\text{V6}(U,U), \text{V6}(U,U), \text{sameTopic}(U, f2(U,U))\)
Q''3(U):
\(\text{V6}(U,U), \text{V6}(U,U), \text{V6}(U,U)\)
simplified in:
Q''4(U):
\(\text{V6}(U,U)\)
=> a valid query plan
Summary

- When the queries and the views are (unions of) conjunctive queries over simple relational schemas, the number of (maximal) conjunctive rewritings is finite and there are several algorithms to compute them.
- It is not necessary the case when constraints are added:
  - to the mediated schema
  - to the views (to express constraints on their access)

DL-Lite (again)

- If the constraints on the schema are expressible in DL-Lite:
  - Consistency checking of the views:
    - Saturation and translation of the NIs into boolean conjunctive queries
    - Application of MiniCon for computing the rewritings of those boolean queries into views
    - Evaluation of those rewritings against the view extensions
  - Rewriting of the query:
    - Reformulation of the query using the PI
    - Application of MiniCon for computing the rewritings of each reformulation
- The computation of all the answers is not possible when the schema constraints requires (slight) extensions:
  - The instance recognition (and thus the tuple recognition problem) is NP-complete in data complexity for slight extensions of DL-Lite
References

- Information Integration Using Logical Views, J. Ullman, Proceedings ICDT '97