Data Integration

Serge Abiteboul Ioana Manolescu Philippe Rigaux
Marie-Christine Rousset Pierre Senellart

Web Data Management and Distribution
http://webdam.inria.fr/textbook

November 17, 2011

Outline

Data Integration

- Goal: obtain data from different data sources with a single query/interface
- Example:
  - Sciences: query different databases recording information about genes
  - Business: query catalogs of different vendors
  - Administration: integrate financial data from different branches
  - Web: find data on a person from many Web sources
- Complex task: to describe possibly complex connection between data sources, use semantics
- Buzz word: semantic Web

Semantics: the glue between sources

- The data sources:
  - have been developed independently
  - are autonomous
  - very heterogeneous
- Semantics is needed to relate their concepts and their structures
- Logic is used to describe the semantics

Example

- Where can I see a film of Woody Allen today in Paris?
  - Woody Allen *plays_in* a film X
  - X *is_shown_at_theater* Y
  - Y *is_located_in* Paris
- Ignore irrevant sources: Air France, etc.
- Find the relevant sources and understand how to use them:
  - IMDB (Internet Movie Database): movies with lots of information on them; provides the list of movies by Woody Allen
  - Allocine: tells where a movie is showing in Paris
- Combine their results

Two main approaches

- Ask queries to a global schema
- To answer, use data over local schemas
- In the two approaches, formulas relate the local schemas to the global schema
- Warehousing approach
  - Global instance is materialized
  - Data is transformed from the local instances and loaded in global instance
  - Queries are evaluated at the global instance
- Mediating approach
  - Global instance is virtual
  - Queries are evaluated using queries to the local instances

Views

- The integration may be seen as a view of the local databases
- A view is a named query that can be re-used in other queries
- Example
  View1(X,Y1,Y2): Flight(X) $\wedge$ DepartureAirport(X,Y1)
  $\wedge$ ArrivalAirport(X,Y2)
  View2(X,Y): Place(X) $\wedge$ Located(X,Y) $\wedge$ Capital(Y)
- Materialized view: computed in advanced and stored
  spirit of warehousing
  - In memory or in a cache
  - Updates are expensive
  - Maintenance: propagate updates to update the view
- Virtual view: on demand
  spirit of mediation
  - Queries are expensive
  - The view is recomputed each time it is used

Data integration in the two approaches

- user query against a global schema that integrates
  - data source 1
  - data source 2
  - a data integration system that integrates
    - data source 3
    - data source 4
    - a data integration system that integrates
  - a data integration system that integrates
    - data source 5
    - data source 6
    - a data integration system that integrates

Two main approaches: comparison

- Warehousing approach
  - Creation: cost of computation and storage
  - Query evaluation is very efficient
  - Updates are costly: need to propagate local instance updates to the warehouse
    Otherwise data are possibly stale
- Mediating approach
  - Creation: no cost
  - Query: cost of reformulation, possibly recomputation, possibly communication
    typically very redundant unless use of caching
  - Updates: no cost
- Standard tradeoff in databases between updates and queries

The mediator approach - details

- Global schema: Define a mediated schema
  - Structured vocabulary serving as a query interface for users queries
  - Typically, one per domain
- Local schemas: Declare a data source
  - Model the content of the source to integrate in terms of the mediated schema
  - Relate the concepts/relations of the source to those of the mediated schema
- Query processing
  - Reformulate and decompose a user's query over the global schema into queries over the local schema that are run at the data sources
  - Combine the answers of local queries to construct the answer to the global query

The use of logic and database technology

- Define a mediated schema
  - A database schema
  - Constraints: first-order-logic formulas
- Declare a data source
  - A source is a database instance
  - Links with mediated schema: first-order-logic formulas
- Queries
  - Expressed as first-order-logic formulas
  - Global query evaluation may use a database engine
  - Each local query evaluation may use a database engine

Outline

Logic and databases

Logic provides a unifying framework for

- asking queries
- specifying the semantics of queries, and
- formalizing fundamental problems in this setting

Queries

- Formulas in first-order logic (FOL)

$$q(\vec{x}) : \exists\, \vec{y}\ \Phi(\vec{x},\vec{y})$$

- $\vec{x}$ is a vector of free (distinguished) variables
- $\Phi(\vec{x},\vec{y})$ is a formula
  - its free variables are those of $\vec{x}$ and $\vec{y}$
  - also possibly constants and bound variables

Semantics in Description Logic

- Given a KB K
  - K is a set of closed formulas
  - An ABox that defines a set of facts
  - A TBox that possibly includes contraints
- Answers to a query q (given K)

$$\mathrm{Ans}(q,K) = \{\ \vec{a} \mid K \models q(\vec{a})\ \}$$

  - $\vec{a}$ is a vector of constants appearing in K
  - $q(\vec{a})$: subtitute in $\exists$ y Φ(x,y) variables in $\vec{x}$ by constants in $\vec{a}$
  - K $\models$ $q(\vec{a})$ if $q(\vec{a})$ holds in all interpretations satisfying K (in all models of K)

Static analysis problems in FOL

- Satisfiability checking of a formula or a set of formulas
  - Is the formula satidfiable?
  - E.g., does there exist of a model?
- Implication of a formula by a set of formulas
  - : $\varphi_1 \wedge ... \wedge \varphi_n \models \varphi$
  - Is it true that each model of φ_1 ..., and φ_n is a model of φ?
  - Can be reduced to satisfiability checking: $\{\varphi_1, ..., \varphi_n, \neg\varphi\}$ is unsatisfiable
- In general, these problems are not decidable (they are recursively enumerable)
  - There does not exist an algorithm to decide whether any formula is has a (finite) model or not
  - One can enumerate the finite interpretations and if one is a model, one will find it
  - Database: finite interpretation; vs. standard logic arbitrary interpretations

Static analysis problem: Query containment

- Le q1($\vec{x}$): $\exists\ \vec{y}1\ \Phi_1(\vec{x},\vec{y}1)$ and q2($\vec{x}$): $\exists\ \vec{y}2\ \Phi_2(\vec{x},\vec{y}2)$
- q1 $\subseteq$ q2 iff
  Ans(q1, I(DB)) $\subseteq$ Ans(q2, I(DB)) for every I(DB)

- Another reasoning problem:
  $\exists\ \vec{y}1\ \Phi_1(\vec{x},\vec{y}) \models \exists\ \vec{y}2\ \Phi_2(\vec{x},\vec{y}2)$?

Conjunctive queries

- Knowledge base is simply a set of facts: a database
- Query: No negation, no disjunction
- Conjunctive query example
  q(X): $\exists$A,C Flight(X) $\wedge$ ArrivalAirport(X,A) $\wedge$ Located(A,C) $\wedge$ Capital(C)
- In datalog notation:
  q(X) $\Leftarrow$ Flight(X) $\wedge$ ArrivalAirport(X,A) $\wedge$ Located(A,C) $\wedge$ Capital(C)

Homomorphism theorem for conjunctive queries

- Suppose to simplify that K is a relational database over a single relation R
- q is a conjunctive query
- K $\models$ q($\vec{a}$) is true iff there exists a homonorphism μ from q to K
  - μ is a function over the variables and constants occurring in q
  - μ is the identity on the constants of K
  - μ maps variables to constants in K
  - for each conjunct R(t1,..., tn) in $\vec{a}$, R(μ(t1), ..., μ(tn)) is in K
- Query evaluation is
  - polynomial in the size of the data (data complexity)
  - NP-complete in the size of the query (program complexity)
  - Data complexity is what matters (the query is typically much smaller)

Homomorphism theorem for containment of conjunctive queries

q1 $\subseteq$ q2 iff there is a homomorphism from q2 to q1

- Algorithm by example
  q1(X): R(X,Y), R(Y,Z), R(Z,Z)
  q2(X'): R(X',Y'), R(Y',Z'1), R(Y,Z'2)
- Freeze the variables X, Y, Z - see them as constants, so q1 as an instance
- Evaluate q2 on the “instance” q1
- If X is an answer
  - then return q1 $\subseteq$ q2
  - otherwise return q1 $\not\subseteq$ q2
- On the example: X is an answer and thus q1 $\subseteq$ q2

No containment: example

- q1(X): R(X,Y), R(Y,Z), R(Z,Z)
  q2(X'): R(X',Y'), R(Y',Z'1), R(Y',Z'2)
- Is it true that q2 $\subseteq$ q1?
- Freezing the variables of q2: X' , Y' , Z'1 , Z'2 are distinct constants
- Evaluate q1 against the instance q2
  - Ans(q1, freeze(q2)) = $\emptyset$
  - thus: q2 $\not\subseteq$ q1

Query containment

- Conjunctive queries: problem is NP-complete
- Central problem for the integration of different data sources
  - Formulas/queries are used to describe data sources
- Other decidable cases:
  - When the queries are expressible in Description Logics
  - Query containment = subsumption between two concept descriptions

Outline

Back to the mediator approach

- user queries
- a mediator system that queries
  - data source 1
  - data source 2
  - data source 3
  - data source 4

Mediation: Underlying principles

- Define a mediated schema (also called a global schema) that serves for the query interface for users
- Declare the data sources: mappings between the global schema and the schemas of the local data sources
- Two approaches:
  - 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
- Query processing
  - Rewriting the users queries (expressed using global relations) in terms of local relations $\Rightarrow$ logical query plans
  - Combine the answers of logical query plans to obtain the result

Different semantics for views

- Consider a view defined by a query def
  $v(\vec{x})$: $def(\vec{x},\vec{y})$
- Exact semantics
  Ext(v) = Ans(def,K)
  axiom: $\forall \vec{x}\ [v(\vec{x}) \Leftrightarrow \exists \vec{y}\ def(\vec{x},\vec{y})]$ added to K
- Sound semantics
  $Ext(\vec{v}) \subseteq Ans(def,K)$
  axiom: $\Rightarrow$
- complete semantics
  $Ans(def,K) \subseteq Ext(v)$
  axiom: $\Leftarrow$

Outline

The Global-As-Views approach: example

- 4 local schemas
- 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)

The global instance is a view of the local instances

University (U): S1.Catalogue(U,P) ∨ S2.Erasmus(N,C,U)
∨ S3.CampusFrance(N',P',U)

MasterStudent (N): S2.Erasmus(N,C,U), S4.Mundus(P,C)
∨ 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)
∨ S3.CampusFrance(N,P,U'),S4.Mundus(P,C')

RegisteredTo(N,U): S3.CampusFrance(N,P,U),

Semantics of GAV mappings

- MasterStudent (N): S2.Erasmus(N,C,U), S4.Mundus(P,C)
  $\vee$ S3.CampusFrance(N,P',U'),S4.Mundus(P',C')
- Exact semantics (typically)
  $\forall$ N [($\exists$C$\exists$U $\exists$P (S2.Erasmus(N,C,U) $\wedge$ S4.Mundus(P,C))
  $\vee$ ($\exists$C'$\exists$U' $\exists$P' (S3.CampusFrance(N,P',U'),S4.Mundus(P',C')))
  $\Leftrightarrow$ MasterStudent (N)]
- Sound semantics
  $\forall$ N [($\exists$C$\exists$U $\exists$P (S2.Erasmus(N,C,U) $\wedge$ S4.Mundus(P,C))
  $\vee$ ($\exists$C'$\exists$U' $\exists$P' (S3.CampusFrance(N,P',U'),S4.Mundus(P',C')))
  $\Rightarrow$ MasterStudent (N)]

Query rewriting by unfolding

- The semantics tells how to populate the global relations
- A logical query plan for a given query is obtained by unfolding each atom in the query
- This leads to conjunctions of disjunctions
- Transform to disjunctions of conjunctions to obtain a set of conjunctive querie
- Example: q(N): MasterStudent(N)
  q(N): S2.Erasmus(N,C,U), S4.Mundus(P,C)
  $\vee$ S3.CampusFrance(N,P',U'),S4.Mundus(P',C')
- Transform:
  q(N): S2.Erasmus(N,C,U), S4.Mundus(P,C)
  q(N): S3.CampusFrance(N,P',U'),S4.Mundus(P',C')

More complex example

- Query: q(x): RegisteredTo(s,x), MasterStudent(s)
- Conjunctive views
  RegisteredTo(N,U): S3.CampusFrance(N,P,U)
  MasterStudent (N): S2.Erasmus(N,C,U), S4.Mundus(P,C)
  MasterStudent (N): S3.CampusFrance(N,P',U'),S4.Mundus(P',C')
- 2 rewritings by unfolding:
  (existential variables in the view bodies are replaced by new variables)
  u1(x):
  S3.CampusFrance(s,v1,x), S2.Erasmus(s,v2,v3),S4.Mundus(v4,v2)
  u2(x):
  S3.CampusFrance(s,v5,x), S3.CampusFrance(s,v6,v7),
  S4.Mundus(v6,v8)

Possibly simplification

- Simplification of u2(x):
  S3.CampusFrance(s,v5,x), S3.CampusFrance(s,v6,v7),
  S4.Mundus(v6,v8)
- by unifying the two first atoms into S3.CampusFrance(s,v6,x) with the substitution s = { v5/v6, v7/x } where v5 and v7 are unbounded existential variables
  $\Rightarrow$ equivalent query expression
- Use the Homomorphism theorem for containment of CQ
- 2 resulting logical query plans:
  u1(x):
  S3.CampusFrance(s,v1,x), S2.Erasmus(s,v2,v3),S4.Mundus(v4,v2)
  u'2(x): S3.CampusFrance(s,v6,x), S4.Mundus(v6,v8)

Results and discussion

- The union $Q(q, G)$ of the logical query plans obtained by unfolding the atoms of a query q using a set G of GAV mappings is complete: for every instance I of the source relations,

$$\text{ans}(q, G \cup I) = \cup_{q \in Q(q,G)} \text{ans}(q, I)$$

- 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 u1 will return an empty set of answers
  - u1(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

$\Rightarrow$ The Local-As-Views (LAV) approach

- The mediated schema is designed to remain stable even when data sources join or leave the integration system
- Only incremental changes for the sources that join/leave with no impact on the rest

Outline

The LAV approach

- Starts with a mediated schema, i.e., a set of global relations
- Example: Global schema
  Student(studentName),..., University(uniName)
  Program(title), MasterProgram(title), Course(code)
  EnrolledInProgram(studentName,title)
  EnrolledInCourse(studentName,code), PartOf(code,title)
  RegisteredTo(studentName, uniName)
  OfferedBy(title, uniName)

LAV mappings: defines the local relations

- Local as views: the local relations are defined as views of the global relations
- Example
  - S1.Catalogue(U,P):
    FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P')
  - S2.Erasmus(S,C,U):
    Student(S), EnrolledInCourse(S,C), PartOf(C,P), OfferedBy(P,U), EuropeanUniversity(U), RegisteredTo(S,U') EuropeanUniversity(U'), U≠U'
  - S3. CampusFrance(S,P,U):
    NonEuropeanStudent(S), EnrolledInProgram(S,P), Program(P), Offeredby(P,U), FrenchUniversity(U), RegisteredTo(S,U)
  - S4.Mundus(P,C):
    MasterProgram(P), OfferedBy(P,U), OfferedBy(P,U'), EuropeanUniversity(U), NonEuropeanUniversity(U), PartOf(C,P)

Semantics of the LAV mappings

- S1.Catalogue(U,P):
  FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P')
- Sound semantics (typically)
  $\forall$U $\forall$P [S1.Catalogue(U,P)
  $\Rightarrow$ $\exists$ P' (FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P'))]
- Exact semantics
  $\forall$U $\forall$P [S1.Catalogue(U,P)
  $\Leftrightarrow$ $\exists$ P' (FrenchUniversity(U), Program(P), OfferedBy(P,U), OffereBy(P',U), MasterProgram(P'))]

LAV vs. GAV

- With GAV: query processing is simple
  - Building the rewriting in LAV requires more work than the simple unfolding of the GAV approach
- With LAV: more flexibility and robustness
  - We can define the global schema without knowing the sources
  - We can define the mapping for one source without knowing the others
  - Allows a fine-grained description of the data sources, and a loose coupling between local and global relations
  - E.g.: 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)
- We will present 3 algorithms
  - Bucket
  - Minicon: an optimization of Bucket
  - Inverse-rules: in the spirit of algorithm for GAV

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) over the global schema (sound semantics)
- output: a set of conjunctive queries over the local relations

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

- q(x): RegisteredTo(s,x), EnrolledInProgram(s,p), MasterProgram(p)
- RegisteredTo(S,U) in the definition of S3.CampuFrance(S,P,U)
- S3. CampusFrance(S,P,U):
  NonEuropeanStudent(S), EnrolledInProgram(S,P),
  Program(P), Offeredby(P,U), FrenchUniversity(U),
  RegisteredTo(S,U)
- Bucket(RegisteredTo(s,x)) = { S3.CampusFrance(s, v1,x) }
- Observe
  s,x are mapped to S,U that are distinguished variables - visible in the local relation

Creation of the buckets - careful

- q(x): RegisteredTo(s,x), EnrolledInProgram(s,p), MasterProgram(p)
- S2.Erasmus(S,C,U):
  Student(S), EnrolledInCourse(S,C), PartOf(C,P),
  OfferedBy(P,U), EuropeanUniversity(U), RegisteredTo(S,U')
  EuropeanUniversity(U'), U$\neq$U'
- RegisteredTo(S,U') is also 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 RegisteredTo(s,x) from S2.Erasmus(s,C,U)
- S2.Erasmus(s,C,U) is not added to Bucket(RegisteredTo(s,x))

Combination of the buckets

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

  Bucket(RegisteredTo(s,x)) = { S3.CampusFrance(s, v1,x) }
  Bucket(EnrolledInProgram(s,p)) = {S3.CampusFrance(s, p,v2) }
  Bucket(MasterProgram(p)) = {S1.Catalogue(v3,v4), S4.Mundus(p,v5) }

- 2 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(NxMxV)
  where N= size of the query, V= number of views, M = size of the views
- N buckets containing each O(MxV) view atoms
- The number of candidate rewritings: $O((MxV)^N)$

Verification of each candidate rewriting

- q(x): RegisteredTo(s,x), EnrolledInProgram(s,p), MasterProgram(p)
- r1(x): S3.CampusFrance(s, v1,x), S3.CampusFrance(s, p,v2),
  S1.Catalogue(v3,v4)
- r1(x) is a valid rewriting
- iff r1(x) together with the LAV mappings logically entail q(x)
- iff the expansion of (r1(x)) is contained in q(x)

Verification by expansion and containment checking

- q(x): RegisteredTo(s,x), EnrolledInProgram(s,p), MasterProgram(p)
- r1(x): S3.CampusFrance(s, v1,x), S3.CampusFrance(s, p,v2),
  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), OffereBy(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: σ(g') = g
  - If an existential variable Y of g appears in other atoms g1, g2, ..., gk of the query
  - If Y' = σ(Y) is also existential in the view definition
    - σ(V) is added to Bucket(g) only if g1, g2, ..., gk are also covered by the definition of σ(V)

Illustration

- V4(X) : cite(X,Y), cite(Y,X)
  V5(X,Y) : sameTopic(X,Y)
  V6(X,Y) : cite(X,Z) , cite(Z,Y) , sameTopic(X,Z)
  Query : Q(U) : cite(U,V) , cite(V,U) , sameTopic(U,V)
  Bucket (cite(U,V))?
    - V4(U) is not added because sameTopic(U,V) is not covered by the definition of V4(U)
    - V6 ?

  σ(X)=U et σ(Z)=V
  Covering of cite(V,U) by the definition of V6(U,Y) $\Rightarrow$ σ(Y)=U
  Covering of sameTopic(U,V) by the definition of s (V6(X,Y))? yes
  Bucket(cite(U,V)) = { V6(U,U) }
  cover(V6(U,U)) = { cite(U,V), cite(V,U), sameTopic(U,V) }
  $\Rightarrow$ r(U): V6(U,U) 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
- Theoretical result: same worst-case complexity as Bucket (exponential in the size of the query)
- Experimental result: scalable when there are many views

The Inverse-rules algorithm

- Principle: The LAV mappings are transformed into GAV mappings (called inverse rules) independently of the query
- To do that, we need to introduce existential variables
  - For the existential variables, we use Skolem terms
  - This keeps track of their provenance
- At query time, the rewritings are obtained by unfolding like in GAV
- The unfolding is a little trickier because of the Skolem terms

Inverse rules: introducing Skolem terms

V4(X): cite(X,Y), cite(Y,X)

V5(X,Y): sameTopic(X,Y)

V6(X,Y): cite(X,Z) , cite(Z,Y) , sameTopic(X,Z)

Result of the inversion of the rule:

cite(X,f1(X)): V4(X)

cite(f1(X),X)): V4(X)

sameTopic(X,Y): V5(X,Y)

cite(X,f2(X,Y)): V6(X,Y)

cite(f2(X,Y),X)): V6(X,Y)

sameTopic(X,f2(X,Y)): V6(X,Y)

Query unfolding

q(U): cite(U,V),cite(V,U),sameTopic(U,V)

σ=X/U, V/f1(U) and rule 1.

q'1(U): V4(U),cite(f1(U),U),sameTopic(U,f1(U))

σ=X/U and rule 2.

q'2(U): V4(U), V4(U),sameTopic(U,f1(U)) and rule 3.

σ=X/U, Y/f1(U)

q'3(U): V4(U), V5(U,f1(U))

Recall

1. cite(X,f1(X)): V4(X)
2. cite(f1(X),X)): V4(X)
3. sameTopic(X,Y): V5(X,Y)
4. cite(X,f2(X,Y)): V6(X,Y)
5. cite(f2(X,Y),X)): V6(X,Y)
6. sameTopic(X,f2(X,Y)): V6(X,Y)

Query unfolding - successful example

The evaluation of this query plan will produce no answer:

There is no way to match V5(U,f1(U)) with a fact V5(a,b) in the data source

Query unfolding (illustration continued)

- q(U): cite(U,V),cite(V,U),sameTopic(U,V)

  σ={ X/U, V/f2(U,Y) } and rule 4.

- q"1(U): V6(U,Y),cite(f2(U,Y),U), sameTopic(U,f2(U,Y))

  σ= { X/U, Y/Y } and rule 5.

- q"2(U): V6(U,U),V6(U,U),sameTopic(U,f2(U,U))

  σ= { X/U, Y/U } and rule 6.

- q"3(U): V6(U,U),V6(U,U), V6(U,U)
- simplified to: q"4(U): V6(U,U) $\Rightarrow$ a valid query plan
- Recall
  1. cite(X,f1(X)): V4(X)
  2. cite(f1(X),X)): V4(X)
  3. sameTopic(X,Y): V5(X,Y)
  4. cite(X,f2(X,Y)): V6(X,Y)
  5. cite(f2(X,Y),X)): V6(X,Y)
  6. sameTopic(X,f2(X,Y)): V6(X,Y)

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